Processing and Analysis of Transient Data from Permanent Down-hole Gauges (PDG)

Xiaogang Li

Submitted for the Degree of Doctor of Philosophy

Heriot-Watt University

Institute of Petroleum Engineering

July 2009

ABSTRACT

The Permanent Downhole Gauge (PDG) can monitor the reservoir in real time over a long period of time. This produces a huge amount of real time data which can potentially provide more information about wells and reservoirs. However, processing large numbers of data and extracting useful information from these data brings new challenges for industry and engineers.

A new workflow for processing the PDG data is proposed in this study. The new approach processes PDG data from the view of gauge, well and reservoir. The gauge information is first filtered with data preprocessing and outlier removal. Then, the well event is identified using an improved wavelet approach. The further processing step of data denoise and data reduction is carried out before analyzing the reservoir information.

The accurate production history is very essential for data analysis. However, the accurate production rate is hard to be acquired. Therefore, a new approach is created to recover flow rate history from the accumulated production and PDG pressure data. This new approach is based on the theory that the relation between production rate and the amplitude of detail coefficient are in direct proportion after wavelet transform.

With accurate pressure and rate data, traditional well testing is applied to analyze the PDG pressure data to get dynamic reservoir parameters. The numerical well testing approach is also carried out to analyze more complex reservoir model with a new toolbox. However, these two approaches all suffer from the nonlinear problem of PDG pressure. So, a dynamic forward modelling approach is proposed to analyze PDG pressure data. The new approach uses the deconvolution method to diagnose the linear region in the nonlinear system. The nonlinear system can be divided into different linear systems which can be analyzed with the numerical well testing approach. Finally, a toolbox which includes a PDG data processing module and PDG data analysis module is designed with Matlab.

DEDICATION

To my wife Qing Yang

ACKNOWLEDGEMENTS

First of all, it is hard to express sufficiently my sincere acknowledgements to my supervisor Doctor Shiyi Zheng who give me the opportunity to start my PhD studies at the Institute of Petroleum Engineering of Heriot Watt University. His guidance, advice and constant encouragement and enthusiasm have been all-important factors in the course of my research. My research work is based on his knowledge, ideas and rigorous judgment. I am also indebted to Shiyi for his extremely patient and careful correction of this thesis.

I am indebted to Conoco-Philips, BG group and Wintershall Holding AG for funding for the project, providing me with living expenses and tuition fees. Conoco-Philips and Edinburgh Petroleum Services (EPS) Ltd. are also to be thanked for providing the field data for the study. Schlumberger and EPS are gratefully thanked for providing with the simulation and pressure analysis software.

Many thanks are dedicated to the academics, research staff, secretaries, librarian and those who helped during the course of this study in the Institute of Petroleum Engineering at Heriot-Watt University. Many thank to the PRIME project group, and to my colleagues Wenbin Xu, Fei Wang and Ludovic Ricard.

Finally, I deeply thank my wife Qing Yang for her love, understanding, patience, support and encouragement during the course of my study, both while we were obliged to live in separate countries, and when she was finally able to join me here. I also thank her for her advice about my research. I also thank my parents, sister, brother and daughter for their help and encouragement.

ACDEMIC REGISTRY
Research Thesis Submission

Name*:*	XIAOGANG LI		
School/PGI:	INSTITUTE OF PETROLEUM ENGINEERING		
Version: *(i.e. First, Resubmission, Final)*	Final	Degree Sought:	Ph.D

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) The thesis embodies the results of my own work and has been composed by myself.
2) Where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons.
3) The thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) My thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require.
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

** Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate*:*		Date:	

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in Academic Registry

Received in the Academic Registry by *(name in capitals):*			
1.1 Method of Submission *(Handed in to Academic Registry; posted through internal/external mail):*			
1.2 E-thesis Submitted			
Signature:		Date:	

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

NOMENCLATURE

Symbols

b	Vector b
c_t	Total compressibility
C	Matrix-valued function of the response coefficients
C_w	Wavelet coefficients
C^n	Space of complex number
d_k	Slope of the interplant
$D_{1,k}$	Detail signal
E	Hilbert Space
Ei(x)	Ei function
$f(t)$	Original signal
F_N	Fourier Matrix
$h_0[n]$	Filter
$h(t)$	Response of the linear system
k	Permeability
LS	Method of least squares
M	Number
N	Number
p_0	Initial pressure
$p_u(t)$	Rate normalized pressure response to system
p_{wf}	Well Bottomhole pressure
Δp	Observed pressure in well test
$P(x)$	Interpolating function
$\dot{Q}(t)$	Well rate derivative
$Q(t)$	Production rate
R^n	Space of real number
r	Radial of well
r_D	Dimensionless Radial
S	Subspace

t_D	Dimensionless Time
t_{total}	Total time of one flow period
$t_{\min}$	Minim time scale
TLS	Method of Total Least Squares
$W(S)$	Decompose the signal S
$W^{-1}(Z)$	Reconstruct the Signal Z
x	Vector
$x[n]$	Original signal
$X(\omega)$	Result of Fourier transform
y	Vector
Y	Decomposed signal
ϕ	Porosity
μ	Viscosity
t	Time
$u(t)$	Unit step function
ε	Measurement errors
δ	Measurement errors
ψ	Wavelet function
$\varphi_k[n]$	Basis functions
λ_o	Outlier Threshold
λ_{n}	Noise threshold
λ_{BU}	Pressure-data slope of BU
λ_{DD}	Pressure-data slope of DD
$\gamma(t)$	Pressure derivative
$\| \ \|_2$	Euclidean length of the vector

LIST OF PUBLICATIONS AND SOFTWARE

1. Software --- PRIME_TOOLBOX is designed to process and analyze the PDG data with Matlab. This toolbox included five sub-modules: data collection, data processing, traditional well testing, numerical well testing and dynamic forward modeling.The data processing module of PrimeToolbox has been tested by India software company Infosys and licensed by the EPS Weatherford.

2. Zheng shiyi and Xiaogang Li., Institute of Petroleum Engineering, Heriot-Watt University, "Analyzing Transient Pressure From Permanent Downhole Gauge (PDG) using wavelet method" ,Paper SPE 107521 presented at the SPE Europec/EAGE Annual Conference and Exhibition held in London, UK, 11-14 June 2007

3. Zheng shiyi and Xiaogang Li, Institute of Petroleum Engineering, Heriot-Watt University, "Transient Pressure analysis of 4D reservoir system response from permanent downhole gauge (PDG) for reservoir monitoring, testing and management", Paper SPE109112 presented at the 2007 SPE Asia Pacific Oil & Gas Conference and Exhibition Held in Jakarta, Indonesia,November 2007

4. Zheng Shi Yi and Li Xiaogang: “Individual well flowing rate recovery from transient pressure with either assigned daily rate or total cumulative production of the well or group of wells through wavelet approach”, in press for publication in Journal of Petroleum Science and Engineering, 2009

CHAPTER 1 INTRODUCTION

1.1 Aim and objectives of study

The petroleum industry has been developing for over one hundred years. The demand for oil and gas are increasing rapidly but new reservoirs are very hard to find and most oil and gas supply is coming from mature reservoirs. Reservoir development involves making many complex decisions based on the reservoir model and reservoir performance and the key to making successful decisions relies on the correct reservoir model. These interrelated tasks can be fulfilled through reservoir description and production history matching. The need for accurate reservoir information first leads to the use of surface gauges to monitor wells. The data most used for history matching in reservoir simulation are surface measurements such as flow rates and cumulative volume of production (oil, gas, and water) and injection (water and gas). Data from surface gauges was not found to be sufficiently accurate for reservoir simulation and so improvements to a reservoir model and its calibration using measured reservoir history requires continuous measurements from the reservoir itself. This finally led to the invention of Permanent Downhole Gauge(PDG), which can monitor the reservoir in real time for a long period of time.

Currently, most oil companies have installed the PDG to obtain downhole data such as pressure, temperature and rate. PDG can record these data in real time during the life of a production well under reservoir conditions. However, this produces a huge amount of real time data, which can potentially provide more information about wells and reservoir. Extracting useful information from the PDG data brings new challenges for the industry and for engineers. Our studies mainly concentrate on the PDG pressure data.

The first objective of this study is to develop a new method for PDG pressure data processing. PDG data is recorded under unconstrained circumstances, which can not avoid noise due to rate fluctuation, and influence from the other wells, as well as workover etc. Therefore, the original PDG data can not be analyzed without processing. The key point of data processing is to identify the transient point of every flow period. A new algorithm based on the wavelet transform theory will be developed for this purpose.

The second target is to analyze PDG pressure data. Before analyzing the PDG pressure data, it is essential to obtain precision rate history. This study will develop a

new approach to reconstructing rate history from cumulative production and PDG pressure. With clean PDG pressure and rate, the PDG pressure data is first analyzed with a traditional well testing approach to obtain the changed reservoir parameter from different Build-Up (BU). The second stage to analyze the PDG data is using numerical well testing. Numerical well testing integrates static geological information and dynamic production information to carry out forward modeling. These two approaches are based on linear systems theory. However, the PDG pressure data is produced from a nonlinear reservoir system which may be caused by changed reservoir parameters, the presence of multi-phases and multi-wells. Therefore the research focus of this study is on extending the current well test method to a dynamic reservoir system using the deconvolution approach.

The final target of the current study is to develop a toolkit which realizes the current research workflow of the proposed methods/approach. The development of this toolkit will be the basis for future research.

1.2 Development of reservoir permanent monitoring equipment

1.2.1 Evolution of the PDG

It is difficult to find who was responsible for the first stand-alone permanent gauge for reservoir and production monitoring and where it was installed, but it seems that it may have been in 1963 in Nebraska, USA (W.A. Nestlerode, 1963 [120]). Several oil companies have explored this area during the 1970s. The first permanent pressure gauge run by Schlumberger was for Elf in Gabon Africa in 1972 was followed one year later by the first North Sea installation on Shell's Auk platform. Another company called Petrobras in Brazil was an early leader, trying a similar gauge system in a subsea well, in the Campos Basin, in 1977. Thereafter, another subsea installation appeared offshore of West Africa in 1978 with the same type being used by Mobil in the North Sea the next year. Up to 1985, a number of US platform wells were installed with permanent P and T (pressure and temperature) gauges wired to the surface by braided logging cables - identical to the cables used for short term wellbore logging. The cables were secured to the tubing by Band-It metal straps, and two-component gauge carriers were used At this stage the devices had very low reliability and problems arose due to loose metal straps falling into the well, gauge carriers twisting and breaking gauges, cables failing when running in or soon after installation, for example (Ziebel,

[125]).

During these early stages, the failure rate of the PDG was too high. Many early failures were caused by damage during installation or by cable problems (H.M Froa et al, 2006 [44]). Later, Shell worked on the development of a downhole electrically operated safety valve. For this, they got a US/Dutch cable manufacturer to develop the first 1/4" tube encased downhole cable. This cable was later brought into use for downhole permanent gauge installations - greatly improving the reliability of the cable part of the systems(Ziebel, [125]).

The first company that saw the potential benefits from long term downhole monitoring was Statoil, and for them the new generation system was developed; tube encased cable with outer plastic protection, cast cross coupling cable protectors, metal sealing between gauge and cable, solid gauge carriers, metal sealing systems for wellhead feed through and a pressure testable wellhead barrier. In addition, a HP computer system was assembled with custom software to sample multiple gauges from one computer system and transfer these data to Statoil's mainframe system. Statoil released publications claiming that the use of permanent downhole P&T gauges increased production by 10% from their platform wells and 15% from their subsea wells (Unneland and Haugland, 1994 [111]).

The experience of Statoil in the application of PDG promoted the development of PDG. The last ten years of the last century was a prosperous time for permanent downhole equipment. The permanent downhole P and T monitoring developed from the electronic P and T gauge system, the fiber optic P and T gauge system to the wireless downhole P and T gauge systems with multi-drop sensors. Permanent downhole P&T monitoring is becoming a standard high production wellbore completion component in the North Sea, Gulf of Mexico, and offshore Brazil and West Australia. In addition to the boom in permanent downhole P and T systems, the requirement also promoted the development of Distribution Temperature System (DTS), downhole flow metering and permanent micro-seismic downhole systems. The first DTS system was installed in a California land well in 1995. The first permanent micro-seismic downhole systems were installed in Oman. The first commercial optical two-phase, non-intrusive, flow meter was installed from the Mars platform in the Gulf of Mexico, USA in 2000(Ziebel [125]).

In recent years, the permanent downhole gauge system tends to integrate all downhole gauges. Low-cost downhole permanent P and T gauge systems have been commonly

used with other downhole systems like DTS, flow meters and micro-seismic downhole sensors. The first fiber optic P and T gauge was installed together with a DTS system in a long reach land well in Southern England in 2001. In the US, Optoplan/Cidra installed the first performwnhole fiber optic Venturi based 3-phase flow meter. The first multizone completion system having fiber optic multidrop P&T gauges and remote operated inflow valves were installed in the US. The first commercial permanent fiber optic seismic sensors together with fiber optic P and T was installed offshore of Norway in an injection well in 2006.

By 2008, worldwide, more than 200 fiber optic P and T gauges and close to 40 downhole fiber optic flow meters and 3 fiber optic seismic downhole sensor systems were now installed. It was estimated that close to 10,000 electronic P and T gauge systems are currently installed, while there must be more than 200 permanent DTS systems. In addition, more than 4,000 fiber optic DTS interventions have been performed. DTS will prove itself as a reliable and powerful flow allocation tool.

1.2.2 Permanent monitoring system

Permanent monitoring systems measure and record well performance and reservoir behavior from sensors placed downhole during production. These measurements give engineers information essential to dynamically manage hydrocarbon assets, allowing them to optimize production techniques, diagnose problems, refine field development and adjust reservoir models. Sensors are placed downhole with the completion string close to the heart of the reservoir. Modern communication facilities provide direct access to sensor measurements from anywhere in the world. Reservoir and well behavior may now be monitored easily in real time, 24 hours a day, throughout the lifetime of the reservoir. Engineers can capture performance daily, examine responses to changes in production or secondary recovery processes and also have a record of events to help diagnose problems and monitor the performance of the well and reservoir.

Permanent monitoring can vary from a simple pressure- and temperature (P and T) sensor, P and T plus vibration monitoring, P and T plus distributed temperature sensing, P and T plus flow measurements, to P and T plus seismic measurements. The permanent downhole system has evolved from the electronic system to the current fiber-optic system. The following part will discuss these two systems, focusing only on the PDG pressure and temperature systems.

Most of the downhole pressure and temperature systems installed today are electronic systems. The various components of a monitoring system are: (a) electronic gauges, (b) gauge mandrels, (c) connectors, (d) cable, (f) data acquisition systems, (g) interpretation software, and (h) power supply.

(a) *Electronic gauges*

Previously, electronic systems utilized strain gauge technology for pressure measurements; this technology did not provide very good resolution. In addition the drifting of strain gauges is higher than that of drifting in quartz gauges. Different types of electronic quartz sensors are used to measure downhole variables, including three types of non-resonant quartz sensors: quartz capacitance sensors, piezoelectric sensors, and vibrating beam sensors. The following table is a summary of performance of different types of quartz sensors.

Sensor Type	Accuracy	Maximum Temperature(C)	Frequency (MHZ)	Pressure Range (psi)	Drift Rate/yr
Piezoelectric Sensors	0.1-0.01%	150	2-10	0-6000	7-17psi
Capacitance Sensors	0.1-0.01%	150	-	0-10,000	7-14psi
Bourperformn Tube Sensors	0.01%	>150	-	0-6000	<8psi
Thickness Shear-Mode sensor	0.01%	175	-	110-16,000	>7psi
Vibrating Beams Sensor	0.0015%	150-175	-	100-15,000	1.5-2%

Table 1.1: Performance of different quartz sensors (Risi, 2004 [93])

Today the resonating quartz sensor (thickness shear mode sensor and vibrating beam sensor) is the primary sensor technology utilized in downhole pressure measurement. Permanent gauges have to stay in wells for several years. Reliability is a key feature and this is inversely proportional to temperature, time and wellbore chemistry.

The permanent quartz gauge measures downhole temperature and pressure. High measurement stability and long life are achieved by using hermetically sealed quartz crystal resonators, digital electronics and proprietary mechanical seals. Figure 1.1 is an illustration of a permanent quartz gauge.

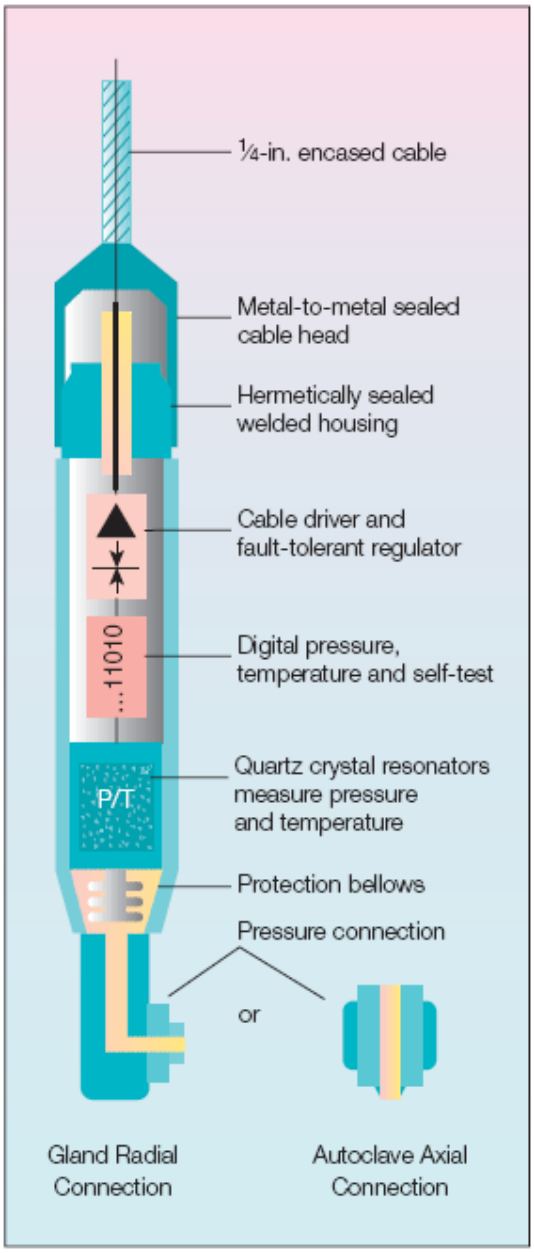


Figure 1.1 Permanent quartz gauge (source: Baker, Jeffrey, Thomas and Unneland , Oilfield Review No 4, Vol.7, Winter,1995)

(b) *Gauge mandrel*

The gauge mandrel serves as a protective housing for electronic gauge. It helps to prevent mechanical damage along the entire length of the gauge. Protection against mechanical damage becomes an event greater issue when the well is deviated. This is because of the presence of liner hangers through which an electronic gauge is moved. Another situation where mechanical damage can be an issue is from a floating vessel. Fig 1.2 is an illustration of how a gauge mandrel protects the electronic gauge.

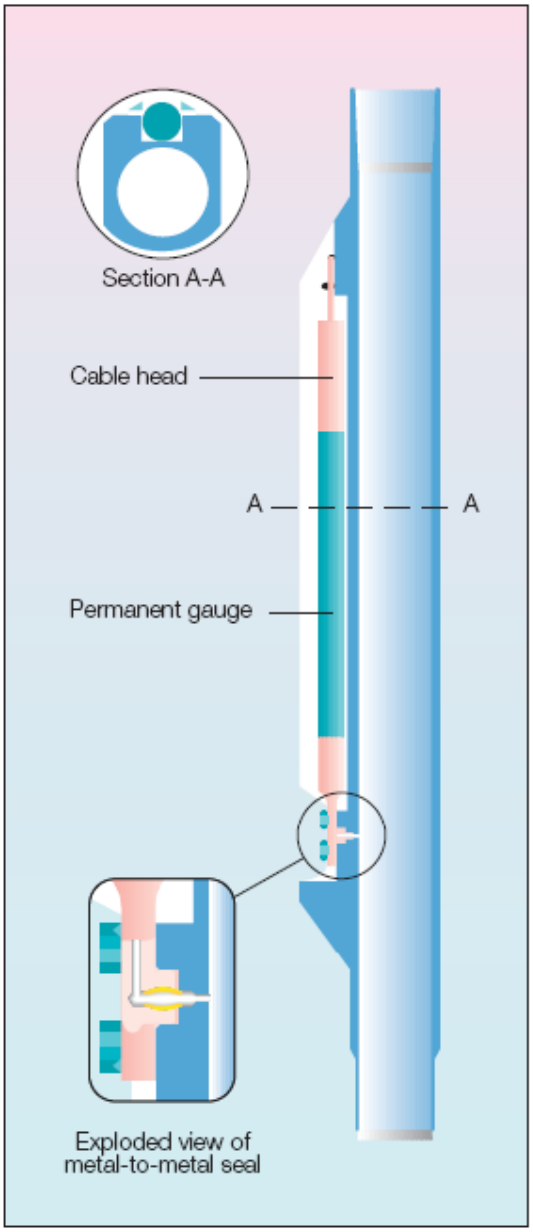


Figure 1.2 Gauge mandrel (source: Baker, Jeffrey, Thomas and Unneland , Oilfield Review No 4, Vol.7, Winter,1995)

(c) *Connectors*

There are two connections to the permanent gauge: electrical connection to the cable for power and data transfer, and hydraulic connection to connect the sensor to tubing under pressure. The electrical connection is usually made at the workshop. The conductor is soldered to the feed-through connector. The pressure connection is made at the wellsite with metal to metal seals. Metal-to-metal seals are also made between the gauge and its gauge carrier or gauge mandrel. At the wellhead end of the cable, metal-to-metal seals are again made to ensure that connections are pressure tight. Each

connection is pressure tested and verified during installation at the wellsite.

(d) *Cables*

Permanent downhole cables have to withstand pressure, temperature and exposure to highly corrosive wellbore fluids during the life of the permanent installation. They also have to be mechanically rugged so that they are not damaged during installation. Cables consist of copper conductors surrounded by Teflon insulation material, anti-slip filler, and standard 1/4-in. stainless-steel or nickel alloy tube and thermoplastic encapsulation material. The filler material supports the cable inside the tube preventing the entire weight of the cable from being supported by the tip connector. It allows some movement inside the stainless-steel tube so that the cable is not exposed to thermal stresses. The metal tube has up to 20,000-psi collapse pressure and prevents wellbore fluid contamination which could short circuit the insulation. Encapsulation helps prevent cable damage such as nicks and crimping during installation. Even so, the cable requires careful handling. Cables usually have single conductors, but can be manufactured with more. Encapsulation materials and sizes can also be tailored to oil company requirements. Figure 1.3 show the evolution of permanent downhole cable.

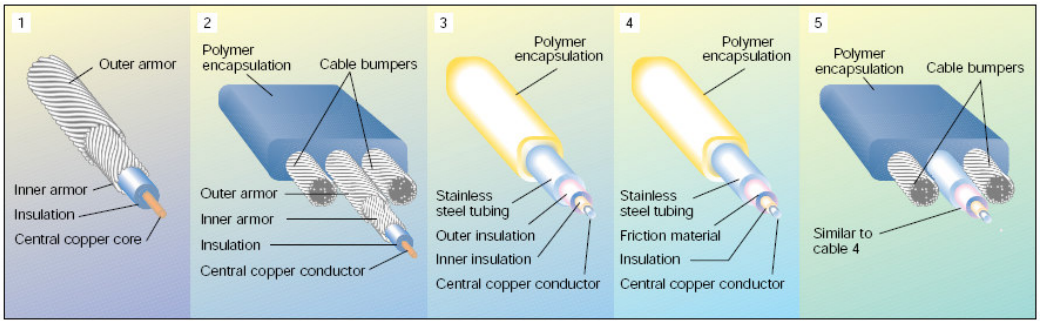


Figure 1.3 Permanent performwnhole cable evolution (source: Alan Baker, John Jeffrey, Alan Thomas and Trond Unneland, Oilfield Review No 4, Vol.7, Winter,1995)

(e) *Acquisition systems*

Various methods are available for collecting data from permanent gauges. In some cases, acquisition systems already in use for collecting data from logging tools are sufficient. It is very common in subsea completions to use an already existing data acquisition system that has been set up to monitor data from the sub-sea wellhead such as surface flow-rates, temperature, pressure, valve positions, and valve status. Connection with an already existing data acquisition system is usually made with a permanent gauge interface card. Interface cards are connected to platforms by cables. Another type of acquisition system can be used such as the hydro-acoustic system. This system does not require a cable to connect the platform.

The Data Acquisition Unit (DAU) records and checks the accuracy of each measurement. A DAU can be interrogated regularly with an acoustic transducer. This acoustic transducer may be attached to the side of the boat, rig, platform, or helicopter. The subsea equipment is powered by a battery pack that can be replaced by divers without losing the DAU memory.

For platforms, several permanent gauges may be connected to an autonomous surface unit that is rack-mounted in the cabin or packaged in an explosion-proof box near the wellhead. This acquires and records the raw measurements and communicates with the oil company's computers via standard modem data links or local area networks. Communication may be via satellite to the oil company office anywhere in the world.

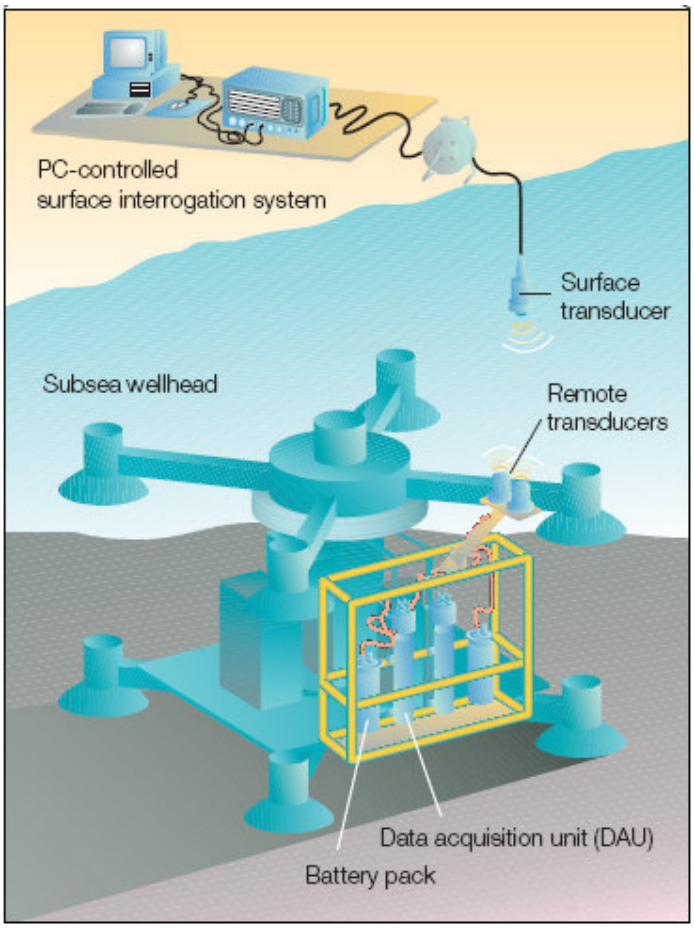


Figure 1.4 Acoustic data link (source: Baker, Jeffrey, Thomas and Unneland, Oilfield Review No 4, Vol.7, Winter,1995)

(g) *Interpretation software*

Permanent gauge monitoring software enables a user to control and monitor permanent gauges from anywhere in the world. This windows-based PC software makes full use of standard communications networks and straightforward point and click menus and

icons. With this software, a user can view the real-time downhole gauge measurements directly or display recorded data files. In addition, the data can be shared via networks with other users for further analysis and interpretation.

(h) *Power supply*

Gauge power is provided from surface directly from subsea umbilical, platform supplies or from subsea battery packs. On land in sunny areas batteries can be recharged using solar panels.

In the last few years, optical fibers have gradually gained acceptance as permanently installed downhole sensors in the petroleum industry. Optical fibers have no moving parts, are immune to electromagnetic radiation, and can be installed in a well after completion.

The attractiveness of optical fibers is their higher reliability due to absence of downhole electronics. Optical fibers are a well-developed, high-bandwidth, low-loss transmission medium and they have very high information transmission rates. Fiber optic sensing systems can be implemented in single-point, multi-point, and continuous sensing configurations. To date, the continuous configurations in wells have only been used for distributed temperature sensing (DTS), although distributed strain measurements are also possible.

A fiber optic monitoring system consists of four major aspects or sub-systems. They are: (a) the instrumentation unit, (b) wellhead outlet and surface cable, (c) in-well cable and connectors, and (d) the sensor assembly.

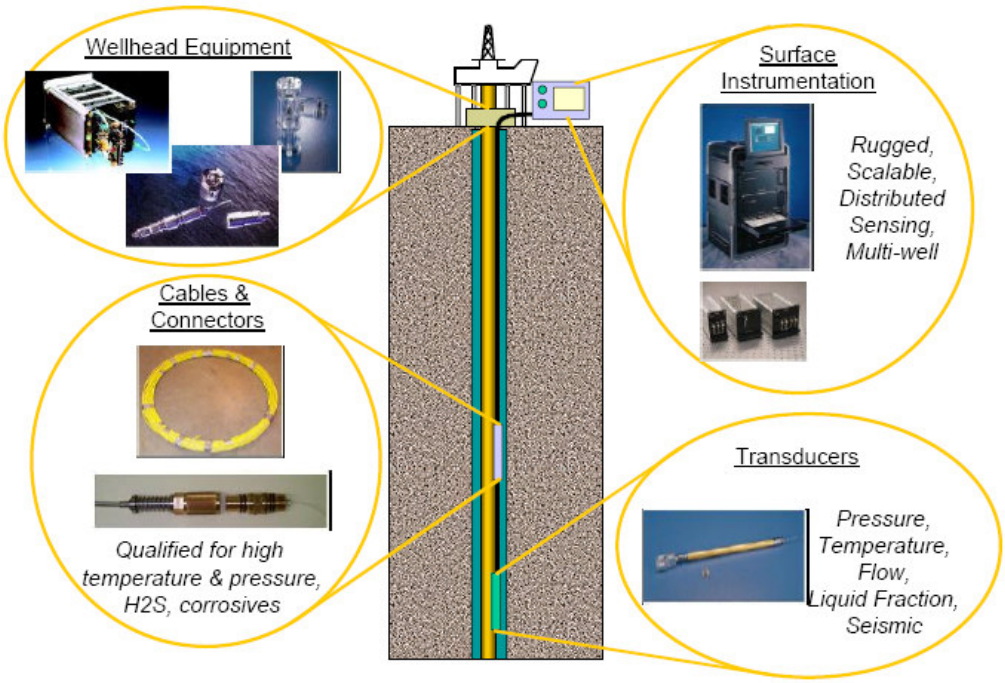


Figure 1.5 Components of a fiber-optic monitoring system (source: SPE71529 [114])

(a) *Instrumentation unit*

The instrumentation unit consists of a fiber optic light source, optic-electronic interrogation unit, signal demodulation unit, microprocessor, monitor, keyboard, associated power supplies, disk drives and data communication interfaces. It also contains the software required to control the data acquisition, conversion, storage and interfacing. In standard implementations, the instrumentation is designed to reside in a control room environment and interface with an external data management system using Modbus protocol or a serial data stream.

(b) *Wellhead outlet and surface cabling*

The wellhead provides for feed-through and exiting of the fiber optic cable from the well in a safe and reliable manner and is similar to that for an electrical system. The standard wellhead outlet contains a minimum of two sealing barriers to every potential leak path and is rated to a working pressure of 15,000 psi. In the case of multi-well installations, multi-core surface cable can be used between the instrumentation unit and junction box while separate cables can be used for connections between each junction box and the well.

(c) *Cable and Connectors*

The in-well fiber optic cable and connector system provides for light transmission to and from the downhole sensors. It is specifically designed for mechanical and

environmental robustness, as well as functional redundancy, and incorporates multiple protective barriers between wellbore fluids and the optical fiber. Mechanical strength and protection of the cable is provided by a 1/4 inch metal capillary tube, encapsulated in a polymeric buffer. The tubing encases a specially coated, small-diameter stainless steel fiber in metal tube surrounded by a buffering material. The optical fibers are packaged in the FIMT with hydrogen gettering grease, which provides high striation forces for holding the fiber in place.

(d) *Sensor Assembly*

The sensor assembly consists of the actual fiber optic sensors and transducers, as well as the mandrel and other equipment required to integrate the assembly into the production tubing string.

1.3 Previous work to process and analyze PDG pressure

1.3.1 Previous workflow

Several authors have suggested a workflow to process and analyze PDG pressure data. The main two workflows are Stanford's (1999 [105]) seven steps workflow and Olsen's two module workflow.

Athichanagorn (1999 [105][106], 2002 [104]) proposed a seven-step procedure for processing and interpreting long-term pressure data from PDG. The seven steps consist of outlier removal, denoising, transient identification, data reduction, flow-history reconstruction, behavioral filtering and moving window analysis. In the first four steps the data is processed using the wavelet approach. The key step of data processing is transient identification. Athichanagorn uses the wavelet modulus maxima to detect the break point of flow period. However, this algorithm can only work for the ideal PDG data. In addition, the analysis part of his workflow analyzes the PDG pressure data using nonlinear regression to obtain the changed reservoir parameters from a simple reservoir model. However this simple model can not represent the true reservoir.

Olsen et al. (2006 [101], 2005 [102][103]) presented an improved workflow involving an automated filter and an automated regression well test module along with a procedure to detect changes or calculate average values of time series parameters. The purpose of the filter module and process monitoring module is to automate filter signals to improve computational performance and to automate the well test analysis for

automatic event detection. The filter module consists of several elements involving outlier removal, denoising and compression. The process monitoring module includes transient detection, slug detection, continuous noise estimation, datum shifting, rate accumulation and detection of shut-in periods. Well tests are best performed through a computer-aided approach using non-linear regression with defined confidence intervals. The analysis module first conducts an automated regression analysis to obtain permeability, skin factor and initial pressure. Then the confidence interval is set to estimate the parameter from non-linear regression. The parameter trend detection tracks the change in parameter mean value and change in parameter variance. The S.Olsen's process modular can identify more phenomena of PDG pressure data. But his algorithm of transient identification has great limitation in processing the real PDG pressure data. And the analysis modular acquires reservoir information from Build Up with the traditional well testing approach. The drawback of this approach is the Draw-Down (DD) of PDG data is not analyzed.

The two workflows descript the big graph of current approach to process and analyze the PDG pressure data. The following two sections will review in detail the current method of data processing and data analysis.

1.3.2 Previous work of processing PDG pressure data

Many papers have discussed PDG data processing since Athichanagorn (1999 [105]) first presented his workflow of data processing. His workflow of data processing includes five steps of outlier removal, transient identification, denoising, data reduction and rate history reconstruction. Outlier removal is the base of the subsequent steps because the outlier has a great effect on transient identification. Transient identification is the key step of data processing because accurate and reliable identification of transient break points in a PDG pressure data is crucial for subsequent interpretation. Data denoising and reduction are necessary steps to obtain clean PDG data Rate history reconstruction is the bridge between the processing and analysis of the pressure data. This section will review these five operations.

Khong (2001 [53]) categorized outliers into step outliers and single spike outliers. A single spike outlier is one data point which lies away from the rest of the data (Athichanagorn, 1999[106]). A single spike outlier arises mainly from a gauge error. Step outliers can be considered as a group of single spike outliers. Step outliers may be created when there is no pressure data available for period of time and zero values or

negative values are assigned (Khong, 2001[53]). Step outliers may also be introduced when interpolation is applied to a data gap.

There are two approaches to remove outliers. The first approach is based on wavelet method (Athichanagorn, 1999[106]; Khong, 2001[53]; Viberti et al. 2007 [23]). Athichanagorn (1999 [105]) found that spike outliers cause discontinuities and create two consecutive singularities in a high frequency data stream after wavelet decomposition. A threshold called the outlier threshold is defined to select data which have two opposite values. The detected position is the spike outlier and is removed from wavelet domain. Khong (2001 [53]) defined step outliers and tried, unsuccessfully, to remove them using Athichanagorn's approach. He therefore, suggested that step outliers have to be removed manually. Viberti et al. (2007 [23]) presented an improved approach, based on the Athichanagorn method. He verified that the detailed signal at the first level of decomposition also shows two peaks of opposite sign every time a pressure transient occurs. As a consequence, the definition of a threshold is not enough to detect all the outliers and to distinguish them from the pressure transients. Therefore, two threshold values must be defined: one of large amplitude (the outlier threshold λ_0) used to identify the two consecutive singularities, and one of smaller amplitude (the noise threshold λ_n), which corresponds to the noise, used to verify whether the singularities in the detailed signal are caused by the outliers. This approach avoids removing some transient data at the beginning of the flow period. However, the single spike outlier in the transient part of flow period can not be identified and removed. And the step outlier is not considered in this algorithm.

The second approach to remove outlier is the median filter approach. Olsen's (2005 [102]) investigation showed that a median filter will work better than the wavelet method for outlier removal. He combined the median filter with accurate noise estimation to remove outliers more efficiently than by a purely wavelet based method. The point that has larger deviation from the trend than the estimated noise level is defined as an outlier. In order to avoid removing any true measurement point, three times the noise level estimate is used for defining the outliers. Thus, if the investigated point has a value three times higher than the median value in the current window, the value is replaced by the median value. The median filter approach can only work when the outlier is located in the stable region of pressure. In addition, this approach is sensitive to the size of selected window and the definition of an outlier. The step outliers are not considered in this approach.

Transient identification also named break point detection or flow event identification is a critical step in PDG data processing. It is used to detect the correct start point of every flow period. Since 1999, different algorithms to achieve transient identification have been proposed. Most attention has been paid to the wavelet method (Athichanagorn, 1999 [106]; Khong, 2001 [53]; Ouyang, 2002 [60]; Rai, 2007 [89]). The wavelet method of wavelet modulus maxima was used for transient detection in 1999 by Athichanagorn. His algorithm uses the spline wavelet and Discrete Wavelet Transform (DWT) to decompose the original signal into different levels of detailed signals and approximation signals. The first step is to determine which level should be used in the detection algorithm based on the assigned resolution of the results. He pointed out the intermediate level of decomposition can normally include the entire transient information. The second step is to choose a slope threshold which will discriminate between singularities corresponding to new transients and noise singularities. The possible location for starting positions of new transients is then detected by wavelet modulus maxima. The exact location of the beginning of a new transient is found by computing the intersection between a line that passes through the first two points in the new transient and another line passing through the last two points in the previous transient (Athichanagorn, 1999 [106]).

Khong (2001 [53]) improved Athichanagorn's method, using Fourier Transform(FT) to screen out false break points with a statistical break point discrimination method. In his study, the real component of the discrete Fourier Transform coefficient's amplitude at both ends of the spectrum is high when the break points are real break points. Conversely, the false break points in the flat region of the transient data have low values of real discrete Fourier Transform coefficients. In addition, he also proposed an approach to adjust the break point using the intersection of two least-square fitted straight lines, one to the left of the true break point and the other to the right of the break point. Ouyang's (2002 [60]) method of transient identification is also based on Athichanagorn's method. In his paper, he mainly discusses how to choose the right transient threshold and gave a formula that may be used to provide a reasonable estimate for the transient threshold for transient identification. Rai (2007 [89]) tested a new wavelet algorithm for identification of break points, based on the stationary Harr wavelet transform method and Stationary Wavelet Transforms (SWT). He found the Harr wavelet is more appropriate for transient identification because the Harr wavelet is discontinuous and thus tends to locate the break points more accurately as compared to

other less compacted regular and orthogonal wavelets. He also pointed out the misalignments between features in the signal which are caused by the DWT. However, SWT can keep all relative proportions of the maxima corresponding to each peak, in different levels of decomposition.

Another approach called transient pattern approach is presented by Thomas (2002 [85]) and Olsen (2005 [102]). They have chosen to provide the system with predefined patterns of the transients, one for pressure Build Up (BU) transients and one for pressure Draw Down (DD) transients. He has chosen to include several points both before and after the transients in the detection pattern, to match the predefined patterns. If all the data points lie in the patterns, the break point can be obtained directly from the patterns. Rai (2007 [89]) introduced a novel pattern recognition approach called the segmentation method. The segmentation method can be outlined in three steps. The first step is to identify what can be called strategic points, by solving a sequence of maximum orthogonal distance problems. To begin with, the first and the last point in the dataset is marked as two strategic points. Then another point is selected whose orthogonal distance from the line segment joining the two strategic points is the greatest. Once a point with the greatest orthogonal distance is identified, it joins the collection of strategic points and becomes an end point for two new line segments, from which a point with the greatest orthogonal distance is chosen. This iterative numerical scheme is performed until the greatest orthogonal distance of data point from the associated line segments falls below a prescribed tolerance. This tolerance can be estimated from the dataset itself. Also, the selection of these points does not require data equally spaced in time.

In addition, Viberti (2005 [36]) utilizes a slope method to detect the break point. The method developed in this work for the automatic detection of transients is based on the analysis of the pressure-data slope. A pressure datum could correspond to the beginning of a new transient if at least one of the following conditions is satisfied:

$$\frac{P_{i+k+1}-P_i}{t_{i+k+1}-t_i} \geq \lambda_{buildup} \qquad \text{for} \quad k=1,2,\cdots,n \tag{1-1}$$

$$\frac{P_{i+k+1}-P_i}{t_{i+k+1}-t_i} \leq \lambda_{DD} \qquad \text{for} \quad k=1,2,\cdots,n \tag{1-2}$$

Where, P_i is the PDG pressure data point; k is the number of data points; λ_{BU} is the minimum slope for Build Up flow period; λ_{DD} is the maximum slope for DD flow

period. The number of control points, nk, can be calculated automatically based on the sampling frequency. The value of λ_{BU} and λ_{DD} depend on the amplitude of the pressure change and need to be calibrated by the person applying the method (Viberti, 2005 [36]).

Another approach, known as Savitzhy-Golay (SG) smoothing filters, was presented by Rai (2005 [90]). SG approach is to first smooth the original signal with polynomial smoothing or least squares smoothing. Filtering data can better preserve the high frequency content of the desired signal. Then, he calculated the derivatives of pressure data. He found there is a sudden change in the gradient of the derivatives data, indicating the vicinity of the break point location.

All these current approaches of transient identification have limitations in processing real PDG data. The transient pattern approach mainly relies on the predefined transient pattern which can only consider an ideal BU and DD and the value of λ_{BU} and λ_{DD} in Viberti's approach depend on the engineer's experience. Both methods are very sensitive to the size of window. In addition, the SG approach has the same drawback as the wavelet approach, although the wavelet method is more flexible. The wavelet modulus maximum is the core of wavelet method. In some real PDG pressure data, we find there is no wavelet modulus maximum. The slope threshold of wavelet method is also very hard to decide. It is therefore necessary to develop a new algorithm to detect the break point.

The noise is a small, high frequency fluctuation in the data and is easily located by the corresponding fluctuations in the detail signals at low levels of decomposition. Several articles present a wavelet method called wavelet shrink to denoise.

Denoising experiments were first run with simulated data to set the threshold level (Kikani, 1998 [54]). Both the soft-threshold and hard-threshold methods proposed are applied to denoise pressure transient data. Athichanagorn et al (1999 [105]) developed a hybrid threshold method for denoising. The hybrid approach considers using the soft-threshold in the continuous data regions and the hard-threshold in the vicinity of the discontinuities. Khong (2001 [53]) proposed the use of a linear interpolation and the calculation of the noise level as a function of the difference between the real pressure data and the interpolated values. The trend of the interval where the noise standard deviation needs to be estimated can be found by locating the slope of the least square error straight line fitted to this interval. The pressure trend is subtracted from the data to obtain the residual noise signal. The standard deviation value is then calculated

from the residual noise signal. Liangbo (2002 [60]) proposed a polynomial method approach to best fit the pressure data for calculating the data noise level. One of the major advantages of the polynomial method is that it may be used for regression for any linear or nonlinear relationships. Olsen (2005 [102]) presented papers which list the result of denoising with different levels, different wavelet type and resolution level. He tried to select the best approach to denoise the PDG data. Viberti (2007 [23]) improved Athichanagorn's hybrid shareholding method with estimations of noise from the first level detail signal.

Although numerous wavelet-based denoising techniques have been presented, several problems still remain before an effective tool can exist. Some wavelet-based techniques have been proven to give good performance in some cases; however, no comprehensive comparison exists to guide the selection of wavelet techniques for PDG data denoising. For most of the proposed techniques several parameters such as noise level, primary resolution level, compression ratio, wavelet type as well as threshold values and the wavelet shrinkage rule need to be determined. In addition, the current approach suffered from the smear effect at the beginning and end of a flow period.

The size of data sets from permanent pressure gauges is enormous. A gauge system with a 1-second recording time interval registers more than 30 million data points a year. Therefore, it is necessary to reduce the number of data points to a data set which is still representative. One of the methods presented for reducing the number of data is the pressure threshold method. In this method, the data are sampled only when a certain change in pressure is reached. However, using a threshold on pressure alone is not sufficient because the pressure may stay relatively constant over a long period. Thus, a time limit should be imposed on the sampling space. The pressure should be recorded when the change in pressure is higher than a maximum preset pressure value and whenever the time span between samples becomes higher than a maximum preset time threshold.

Flow history reconstruction is extremely important for detailed well/reservoir parameter calculations. Several correlation-based and physics-based rate allocation methods have been proposed. The simplest method is to use productivity function or an infinite radial flow model to be applied for flow rate reconstruction. Significant improvement was made by using the exact interpretation model to adjust the flow rate to the corresponding pressure value. But this method has limitations when applied in unstable states and in some complex reservoir models.

Athichanagom (1999 [106]) presented a method to reconstruct unknown and uncertain flow rates during regression on pressure by parameterizing them as unknown parameters constrained to existing rate measurements and production data. The objective function consists of the reduction in error between the real and estimated cumulative production after the rate history has been adjusted. However, this method is based on the reservoir model which we can not confirm from the model. Another approach is based on the application of soft computing (Fuzzy Logic) to investigate the pattern of relationships between the production contribution of layers in commingled reservoir and rock petrophysical data as well as other relevant geological/engineering data. Fuzzy Logic is a qualitative method which largely depends on experiments in reservoir engineering.

Several papers have been published applying the permanent downhole pressure gauges for production or injection profiling and allocation. Sun (2006 [57]) proposed a technique of integrating down-hole real time pressure and temperature data to predict and allocate multiphase production in a multiple zone intelligent well system which used customized Interval Control Valves(ICV) choke performance models. The current method for rate allocation using Permanent Downhole Pressures is proposed by McCrachen and Chorneyko, ExxonMobil (2006 [67]) Upstream Research. This approach includes a four step method to allocate rate by using the permanent downhole pressure. The first step is to determine an initial estimate, which is given by applying allocation factors based on infrequent surface well tests to the total measured production for the rates. Secondly, the measured downhole pressure is analyzed using pressure transient analysis and the initial rates. A simple reservoir model is then developed using the pressure transient analysis and downhole pressures as history-matching criteria. In the third step, this model is used to predict rates using the downhole pressure measurements as an input. The result is flow rates that are consistent with the downhole pressure and are related to the drawdown. These flow rates are predicted using high frequency pressure data. However, these predicted rates do not necessarily match the cumulative production. Finally, an algorithm was developed to provide a mechanism for comparing predicted production rates with the measured cumulative rate and to reallocate the production so that the measured cumulative production is honored. There are two sources of error in this method. First, the difference between the predicted production rate and the real measured cumulative rate may be caused by an inaccurate and simplified model. The authors tried to eliminate the difference by

merely reallocating the rate, without considering changing the model. The other problem is considering the relationship between pressure and rate by a simple model without any hypothesis.

1.3.3 Previous work on PDG pressure data analysis

A wide range of applications of permanent downhole gauge data has been reported in the industry. These applications include: reducing ambiguity and uncertainties in the interpretation; detecting the changes in reservoir properties; monitoring skin, permeability, pressure drawdown over time; evaluating the performance of stimulation or work over; evaluating completion performance; identifying well problems and reservoir connectivity; evaluating operational efficiency; improving flowback time of new wells; assisting reservoir simulation and history matching. Considering these ways in which PDGs are used, one of the most defining characteristics is the inherent combination of short-term and long-term transients. These transients permit dual application of these data to infer short-term effects (e.g., skin effect) and longer-term effects such as pressure depletion or changes in reservoir mechanism. With respect to the longer-term changes, PDG data are especially useful for history matching (Horne, 2007 [95]).

As we review the applications of PDG data, the following section will consider the analysis of PDG in three directions: analyzing PDG short-term characteristics; analyzing PDG long term characteristics and analyzing PDG pressure data with deconvolution.

The PDG pressure data consists of hundreds of BUs and DDs. It is easy to persuade the user to consider using the well testing theory to analyze PDG data. Several authors have published works on the application of PDG pressure data in field cases. Shepherd (1991 [20]) presented the use and application of PDG in the Balmoral Field to monitor Bottomhole Pressure (BHP) during extended well tests of two satellite fields for appraisal purposes. Unneland and Haugland (1994 [110]) presented studies on how Statoil used the permanent downhole gauges for reservoir management of complex North Sea oil fields. For example, the Gullfaks field is heavily faulted with a number of sealing or partially sealing faults, hence one important reservoir monitoring objective is to identify the degree of communication between the fault blocks. Permanent gauges are installed in dedicated wells to monitor the two reservoirs independently.

Athichanagorn (1999 [106]) published a moving window technique to obtain the

dynamic reservoir properties. His approach first chooses a window size and then runs regression on the selected data window to determine the reservoir model parameter. This workflow is then repeated for the subsequent data. Hddad and Proano (2004 [97]) investigated the effects of several reservoir dynamic factors including changes in reservoir pressure, skin, permeability thickness and reservoir drive mechanism from PDG pressure in two fields in the Gulf of Mexico area using traditional well testing. With respect to reservoir description, a few examples have been published in which data recorded by the permanent downhole gauges have been used for multi-well interference tests. Laws (2005 [78]) presented a field case study in the Harweel Cluster in Southern Oman. The PDG was placed in every well to perform an interference test which could provide quantitative estimates of average flow capacity and storage capacity among the wells.

Olsen (2006 [101]) proposed a workflow for completely automated well test analysis. The procedure involves an automated filter and an automated regression well test module along with a procedure to detect changes or calculate average values of time series parameters. The well test module is performed through a computer aided approach using non-linear regression with confidence intervals. The non-linear regression technique provides consistent parameter estimation. All equations are given in Laplace space and inverted numerical solution using the Stehfest routine. The regression is performed using a Levenberg-Margquard optimization routine with analytical derivatives. Confidence intervals are calculated from each of the unknowns.

The PDG pressures also have long term characteristics which several authors have suggested can be integrated with history matching. Faskhoodi (2007 [65]) used low frequency observation data in automatic history matching. His study proposed a technique to reduce the number of observed data points from PDG, so that history matching can be accomplished more efficiently. The wavelet transform is applied to filter the low frequency information. He also demonstrated that the quality of history match is improved by using low frequency data.

Decline-curve analysis is one of the most widely used and documented methods for using long term data for reserve estimation and production forecasting for a field under depletion. Kuchuk (2005 [56]) presented a procedure for applying continuous downhole pressure and rate measurements for decline curve analysis. The convolution of a constant pressure model extends the classic decline-curve analysis method to the cases of wells operated at varying pressures, and allows reservoir description and well

performance monitoring to be performed.

In order to use the dual characteristics of PDG data, there is also another way to analyze PDG pressure. Deconvolution can bridge the gap between the short-term transient and long term trends. Deconvolution is a term used to describe the mathematical manipulation of a complex signal containing complex variable-rate transients, and then to extract from it the underlying simple reservoir response to a single transient. Deconvolution has been used for several decades, for example van Everdingen and Hurst (1949 [2]). The hope of extracting global reservoir information from complex everyday pressure signals has sustained many studies on the subject. In spite of this level of interest, there has been only modest effective success, until recently. PDG data represents a new target for deconvolution, in that the measurements are of very long duration. It would be very attractive to be able to take 2 years of data, which might perhaps be made up of a multitude of transients, each of a few days duration, and to deconstruct the signal to visualize a single reservoir-model response covering the whole 2 years. With the increasing availability of this new form of data, researchers have revisited the topic of deconvolution in recent years. Importantly, a series of studies by von Schroeter et al. (2004 [110]) achieved practical success with deconvolution that had eluded earlier studies.

Deconvolution allows an expanded view of the reservoir behavior in time and, therefore, provides considerable added value in the context of PDG data compared to a simple analysis of only the individual transients. Deconvolution is not without difficulties, however. There is an underlying assumption that the reservoir model remains constant throughout the period of the computation. In practice, this assumption is not always valid. One way that the model may change is if the reservoir parameters alter during the extended period of the PDG record, for example, if the permeability and skin do not remain constant. Another difficulty may occur if differences occur during the buildup and the drawdown periods within the continuous record. Differences between buildups and drawdowns are not necessarily unusual, even in short-term wireline well tests, and have been discussed specifically in the context of PDG data by several authors. Levitan (2005 [69]) showed advantages of focusing attention on only the buildup sections of the data, an approach also favored by Olsen and Nordtvedt (2006 [101]).

1.4 New workflow and organization of thesis

1.4.1 New workflow for data processing and analyzing

Considering the nature of PDG data, changes registered by PDG are subject to far less control than the changes imposed during a conventional well test. The unrestrained fluctuations in well condition measured by a PDG are much more complex in character, and require different analysis tools for interpreting the resulting data (Horner, 2007[95]). This is because the PDG data will contain all information including gauge error and noise, work over information, wellbore information, rate change, reservoir information, reservoir boundary and the influence from other wells. So, the first step in processing the PDG data is to identify the different events, recorded from PDG and to discover what has actually happened in the reservoir and well. These different events are therefore separated to obtain clean data for the further analysis. Transient pressure can be treated as a continuous signal, and signal decomposition can be conducted with the wavelet transform, which can divide the original signal into different frequencies. This signal transform process can help engineer to identify where the event happens. The new approach first decomposes the original PDG pressure signal into different frequency sub-band signals with wavelets. Then the different frequency sub-bands signal will be linked with the event because some reservoir events are stimulated within the specific sub-signal signal. The new workflow for processing PDG signal includes several steps:

I. Data preprocessing
II. outlier removal
III. Flow event detection
IV. Identification of BU and DD
V. Denoising and data reduction
VI. Identifying abnormal events

As pointed out above, one of the more defining characteristics is the inherent combination of short-term and long-term transients (Horner, 2007[95]). The character of the short transients can be analyzed by the normal well test analysis to obtain the dynamic reservoir parameter. However, traditional well testing cannot reflect the complex and heterogeneous reservoir model. Therefore, numerical well testing is introduced for updating the reservoir model. Nevertheless, both the previous two methods are mainly focused on the short term information from PDG data. Currently the deconvolution approach can extend the traditional single flow period well test to a multi-rate flow period, avoiding the effect of changed rates. However, the theory of deconvolution is based on the assumption of a linear reservoir system. Using

deconvolution is impossible for long term PDG data which only record the information from a linear system, because the real reservoir is complex nonlinear system. So, the key step is to linearize the nonlinear reservoir system. These phenomena, which the deconvolution can make mistake when it is applied for the nonlinear system, are used to diagnose the nonlinear system. Therefore, the nonlinear reservoir system can be expressed as a piecewise linear system. Then, numerical well testing can be applied for each linear system to realize dynamic forward modeling. Figure 1.6 shows the overall workflow for processing and analysis of the PDG data as described above.

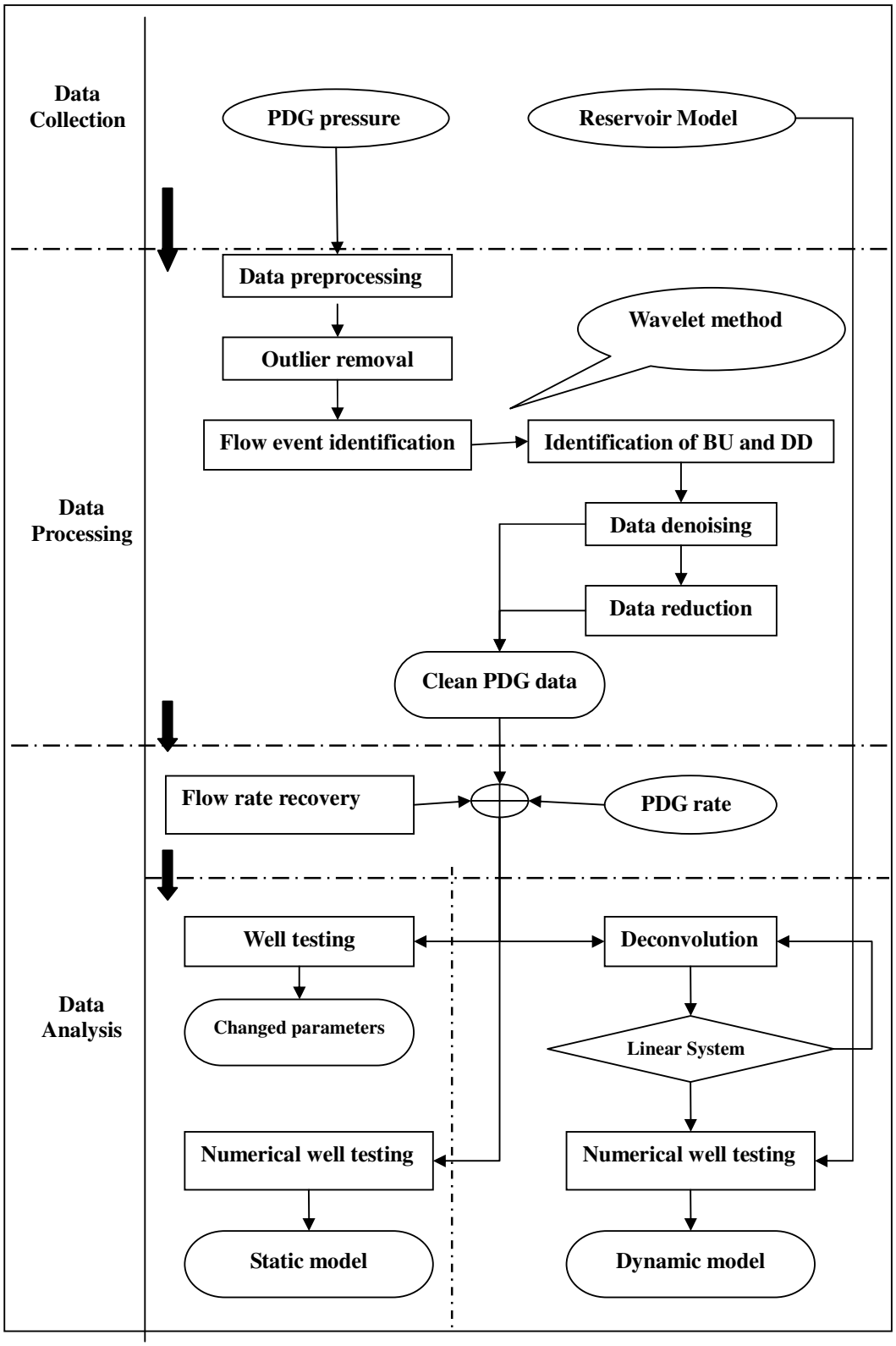


Figure 1.6 Workflow of PDG data processing and analysis

1.4.2 Organization of thesis

This dissertation consists of five chapters. Following the introduction, the aim of this study and the outline of this thesis, Chapter one has reviewed the literature of PDG. The literature review gave a brief overview of the evolution of permanent downhole hardware equipment focusing on permanent downhole pressure equipment. Then, all the methods for processing PDG pressure data were critically reviewed. And the different approaches to PDG data analysis investigated. After reviewing the literature, a new workflow for processing and analysis the PDG data was developed.

Chapter two will firstly discuss the wavelet theory, used for data processing. The wavelet transform can decompose the original signal into different level frequency signals. The wavelet transforms can be applied to process the PDG pressure data because the PDG data includes different levels of information from gauge, reservoir and wells. The main target of PDG data processing is to filter the different noise information from PDG data to keep the real reservoir information.

Chapter three first concentrates on reconstructing the precise rate history of wells. A new approach is introduced to recover production history form PDG pressure and cumulative production. So far, clean and precise rate and pressure data has been achieved. The later part focuses on PDG pressure analysis. The first step is to analyze the BU data from PDG pressure data using the traditional well testing approach. In the second step, the traditional approach is extended to an advanced method. The advanced approach considers the BU and DD as a whole part with the numerical well testing.

Chapter four illustrates how to process the PDG data allowing for the dynamic nature of the reservoir. The new approach to analyzing the dynamic reservoir system integrates well testing, deconvolution and numerical well testing. This chapter first discusses the deconvolution algorithm. Then the second part will introduce how to diagnose a nonlinear system with the deconvolution approach. Using this method, the dynamic reservoir can be subdivided into different linear systems. And the numerical well testing applied for each linear system in order to validate the reservoir model and obtain the correct and final updated reservoir model.

In chapter five conclusions are drawn from the study. Finally section 5.2 on future work introduces idea for further analyzing the PDG data.

1.5 Chapter conclusion

During the last twenty years, PDG equipment has developed from electrical to optical, and then to wireless systems. The data recorded include pressure, temperature, and distribution temperature and production rate. More and more companies have installed PDGs to collect data. However, how to process and analyze such a huge amount of data leads to bottlenecks for the industry and its engineers.

Hundreds of papers have been published to investigate the benefits of PDG data. Although the wavelet method is accepted as the most useful method in processing PDG data, there is still no clear workflow for data processing and some key problems have not been solved. Current approaches can not remove the step outlier and identify the right breakpoint when there is no local modular maximum. In addition, there is still no approach to detect some of the abnormal phenomena encountered in real PDG data.

The first direction is to analyze the PDG data with the traditional well testing methods to obtain the changes in reservoir parameters. The second direction is to use the long term characteristics of PDG data in history matching for reservoir simulation. The third direction is to treat the PDG data as the whole dataset, using the deconvolution method. However, these three approaches are all based on the linear system theory; this means it is necessary to find a new approach to do the analysis of PDG data.

CHAPTER 2 PDG DATA PROCESSING

2.1 Introduction

PDG pressure data can be much like those collected in a conventional wireline test, such as a buildup or a drawdown test. But, there are some important differences between PDG data and conventional wireline well-test data. A conventional well test is designed to record the imposed change in the flow rate in as simple a way as possible. However, the PDG data are more natural than traditional data, as the changes recorded by PDG are subject to far less control than the changes imposed during a conventional well test. The pressure data collected from PDG represents essential information for understanding the dynamic behavior of the field, because PDG can record data during the whole life of the well. Therefore, the unrestrained fluctuations measured by a typical PDG are much more complex in character. This complex, noisy and huge amount of PDG pressure data can not be analyzed without considerable processing.

Several papers presented different workflows for processing PDG pressure data using a wavelet approach. In spite of the many advantages reported from the current wavelet approach, several important issues remain to block the development of PDG data processing. These issues include: detecting breakpoints of PDG data into relevant subsections, identification of abnormal behavior, outlier removal and speed of algorithm.

This chapter will introduce an improved wavelet approach for PDG data processing. A new workflow of PDG data processing is introduced. The background theory will be discussed before data processing. This new workflow first conducts data preprocessing to convert the original PDG data to an available dataset. Then the outliers are removed and the flow events are identified. After that, we focus on the details of a single flow period to detect the BU and DD and some particular phenomena which are recorded in PDG data. Finally, the data denoising and data reduction are discussed for the whole dataset or for a short dataset.

2.2 Workflow for PDG data processing

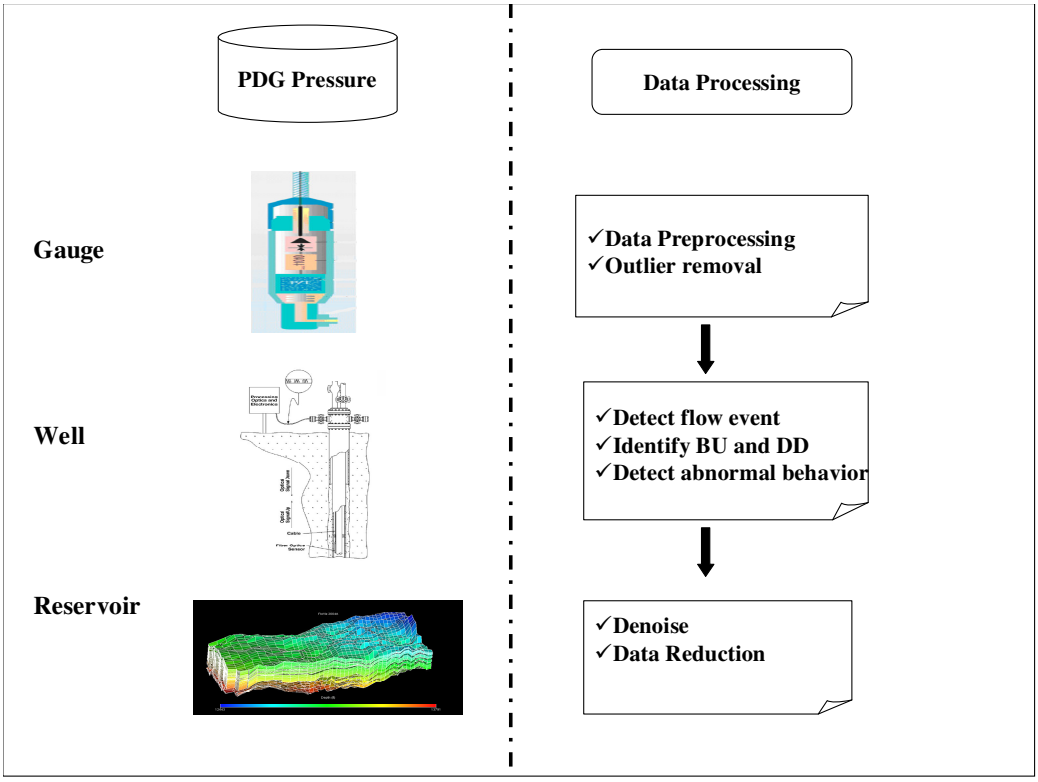


Figure 2.1 The framework of PDG data processing

The PDG data is recorded under the natural downhole conditions of the reservoir. So the PDG data integrate different effects of the environment. From the left part of Figure 2.1, we found the PDG data information can be derived from gauges, wells and reservoir. According to the experience of data processing, different effects of the environment present different characteristics on the PDG pressure data. The gauge may cause phenomena such as negative values, data shifting, data gaps and data overlap. Some high frequency signals in the PDG data may arise from well events such as opening the well, shutting in the well, workover, wellbore storage and phase changes. The low frequency signal, on the other hand, may come from reservoir information including formation properties, the reservoir boundary and interference from other wells. It is impossible to directly obtain useful information from PDG data because the PDG pressure data is a kind of integrative information. Consequently, it is necessary to process the PDG data before the analysis can be performed.

The idea of new data processing workflow is arranged according to the original of PDG data information. The new workflow consists of three modules which is illustrate in the right part of Figure 2.1. The first module is to identify and remove gauge influence

such as outliers, negative values and data drifts. The second module is to identify the flow events of well. The final stage of data processing is to perform data denoising and data reduction before analying the reservoir information.

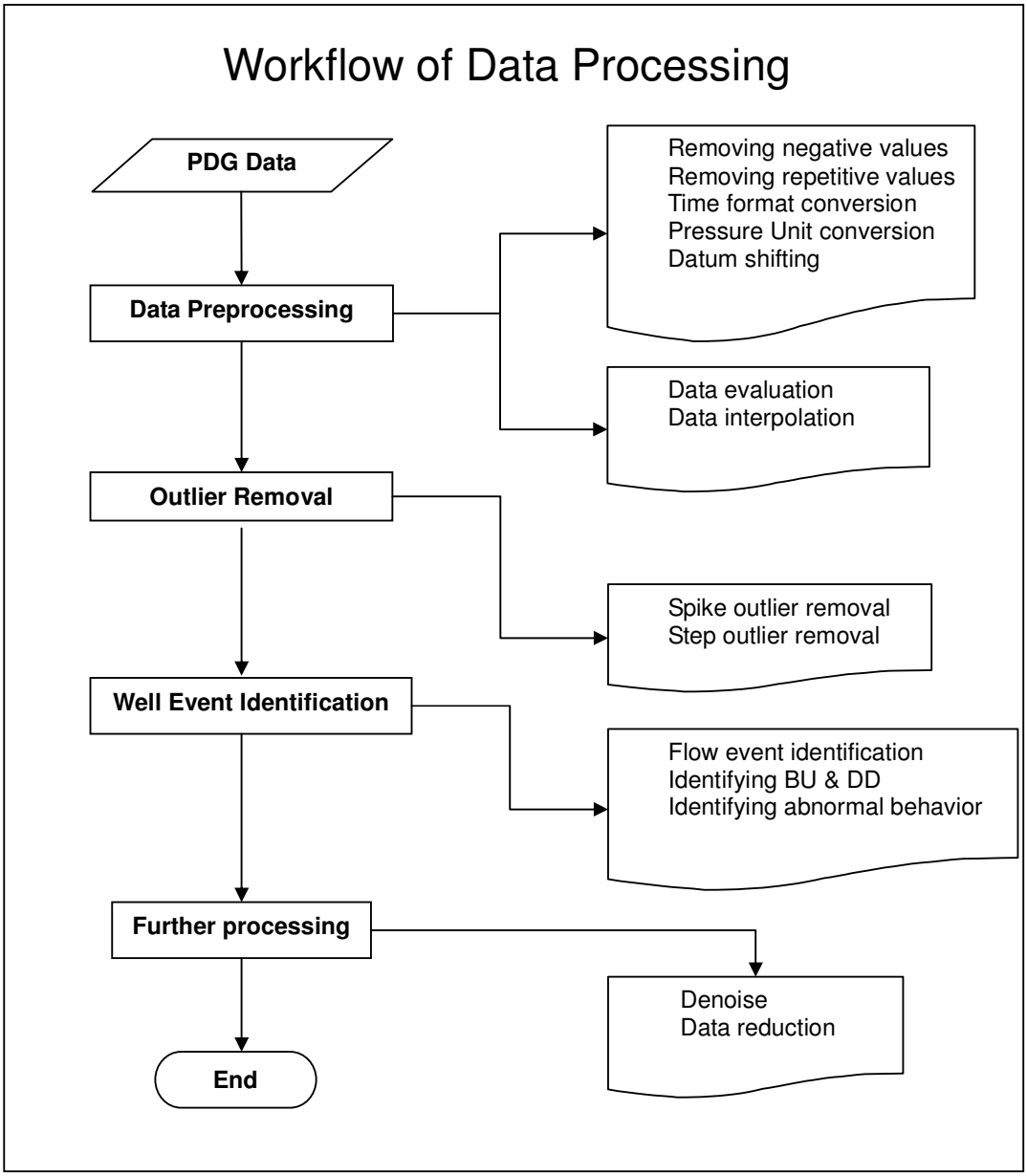


Figure 2.2 The workflow of PDG data processing

Figure 2.2 illustrate the detail workflow of data processing. The new workflow of data processing is based on the wavelet approach because wavelet transform can decompose the original signal into different frequency signasl so as to obtain the different behavior of events for corresponding information.

The first step of data processing is to carry out data preprocessing. The data preprocessing first focuses on removing the negative values and time repetitive values,

converting the time format and pressure format and shifting datum in order to reduce the logical errors and convert the original PDG data format into familiar format. The second step of data preprocessing prepare the basic information of PDG pressure for the subsequence steps. Data evaluation is to obtain the recording frequency of PDG data. The interpolation of the PDG data is to aquire equal time interval pressure data because wavelet transform need the equal time interval pressure data.

The second step of data processing is to filter the effect of gauge with outliter removal. The outlier information are always located in very high frequency signal after wavelet transform. This step can remove two types of outliers: spike outliers and step outliers.

After outlier removal, the PDG pressure information mainly consists of the information from well and reservoir. The well event identification first detect the location of the flow event in PDG pressure data. The whole PDG pressure data can be divided into different flow period with different flow rate. Every flow period is classified into drawdown, shut-in buildup and rate-drop buildup because we will analyze the PDG pressure from the view of well testing. After identification of flow period, we concentrate on the behavior of every flow period because the real behavior of flow period is more complex than the ideal BU and DD. Some abnormal behavior may be caused by artificial factor or other envoirnment factor which can not be simulated or ananlyzed with the normal well and reservoir model.

After well event identification, The PDG data mainly consist of the useful information of well and reservoir. But it still suffers from the noise of pressure data and the size of the data. So, the final stage of data processing is to perform data denoising and data reduction.

2.3 Wavelet theory

2.3.1 Signal analysis

PDG data consists of very complex signals which contain different frequency signals from different part of well and reservoir system. As we observed, the well and gauge information are mainly in the form of high frequency information. The reservoir information is mainly recorded at low frequency. Therefore, to choose the right signal analysis approach is critical step of data processing. Figure 2.3 illustrate three different signal analysis approaches for data processing. The subsequence part will discuss which one we choose to do data processing.

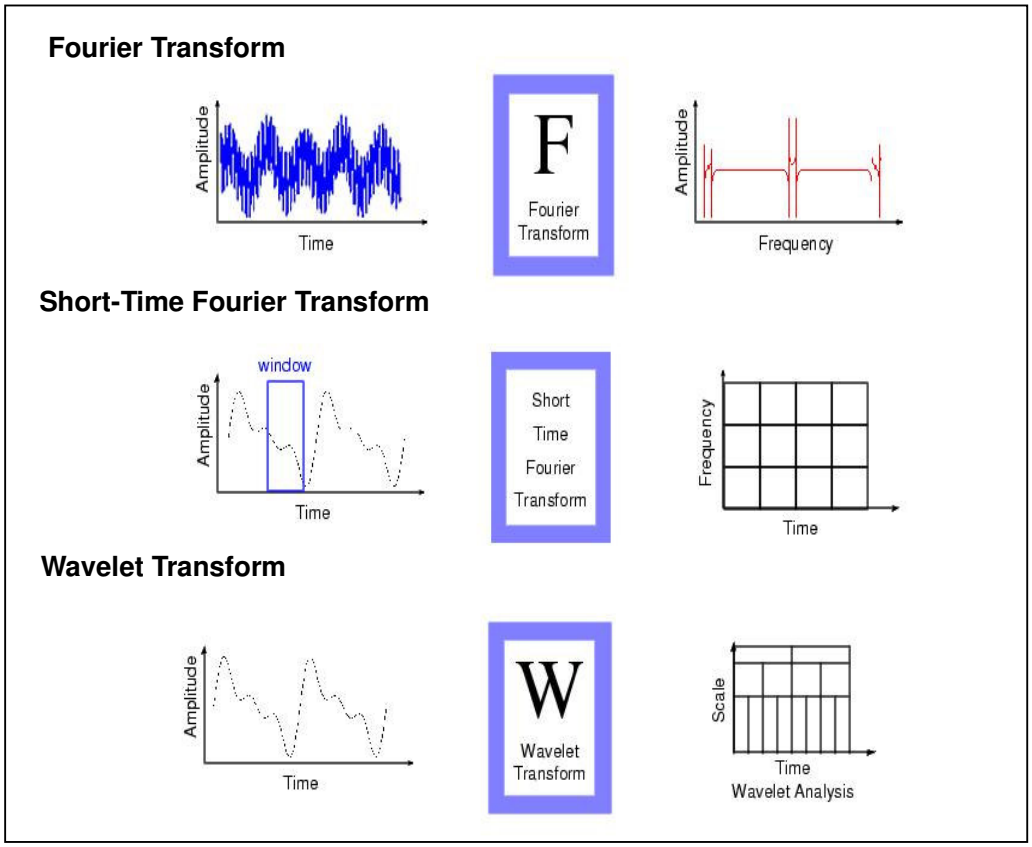


Figure 2.3 Comparison of different signal analysis approach: Fourier Transform Short-Time Fourier Transform and Wavelet Transform (source: Matlab\wavelet)

Fourier analysis can breaks down a signal into different frequencies sinusoids. This transform can change view of signal from time-domain to frequency-domain. From the first part of Figure 2.3, the integrated sinusoids signals present variation in fixed frequency after Fourier Transform. So, the Fourier Transform have serious disadvantage in detect the event of signal because the time information of signal is lost after Fourier Transform. In an effort to correct this deficiency, a technique called Short-Time Fourier Transform(STFT) have been developed to only apply the Fourier transform for a small section of the signal at a time. This transform maps a signal into a two-dimensional function of time and frequency in the second part of Figure 2.3. The STFT have the advantage of providing information about both when and at what frequencies a signal event occurs. However, the problem of this approach is that once you chose a particular size for the time window, that window is same for all frequencies. Many signals require a more flexible approach one where we can vary the window size to determine more accurately either time or frequency. This requirement drives the creation of Wavelet Transform in the third part of Figure 2.3. Wavelet analysis allows the use of long time intervals where we want more precise low-frequency information,

and shorter regions where we want high-frequency information. As we know, the PDG pressure signal consists of some stationary signal from reservoir or well and some transitory signal like drift, outlier, abrupt changes and beginnings and ends of events from gauges or wells. The Wavelet Transform is considered as the proper tools to filter signal from different frequency and different regions.

2.3.2 Wavelet Transform

Wavelet analysis is the breaking up of a signal into shifted and scaled versions of the original (or mother) wavelet. Wavelet function:

$$\Psi_{a,b}(t) = |a|^{-1/2} \Psi(\frac{t-b}{a}) \tag{2-1}$$

The continuous wavelet transform (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function ψ:

$$C(scale, position) = \int_{-\infty}^{\infty} f(t)\Psi(scale, position, t)dt \tag{2-2}$$

The results of the CWT are many wavelet coefficients C, which are a function of scale and position. Scaling a wavelet simply means stretching (or compressing) the wavelet. It turns out that if we choose scales and positions based on powers of two, so-called dyadic scales and positions , then our analysis will be much more efficient and just as accurate. We obtain such an analysis from the discrete wavelet transform (DWT). An efficient way to implement this scheme, using filters, was developed in 1988 by Mallat. The Mallat algorithm is, in fact, a classical scheme known in the signal processing community as a two-channel sub-band coder. Figure 2.4 illustrates the process of decomposition with Mallat algorithm.

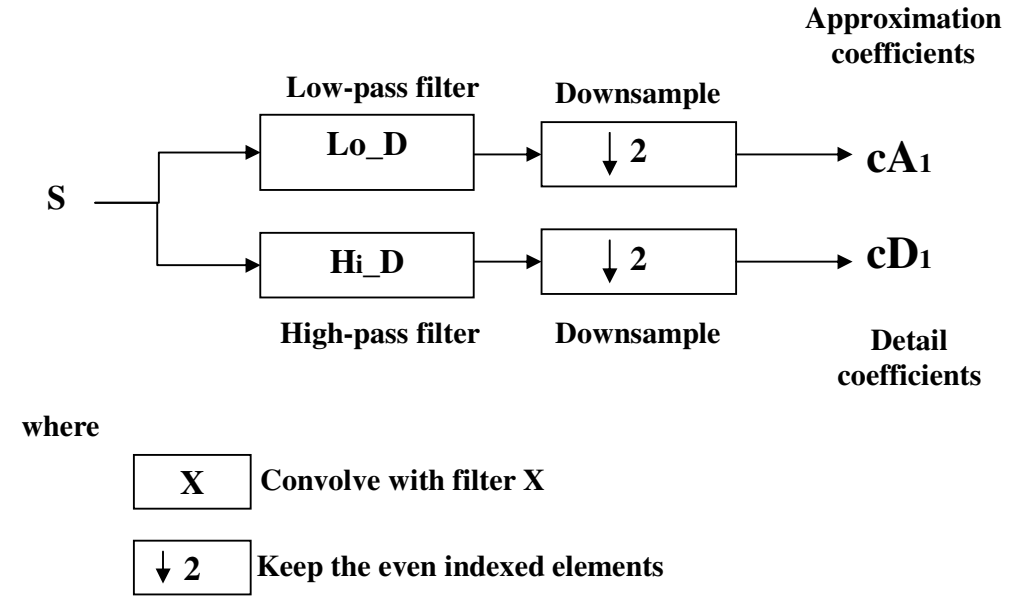


Figure 2.4 Mallat decomposition algorithm (source: Matlab\wavelet)

The next step splits the approximation coefficients cA1 into two parts, using the same scheme, replacing s by cA1 and producing cA2 and cD2, and so on. Figure 2.5 show the multi-decomposition algorithm.

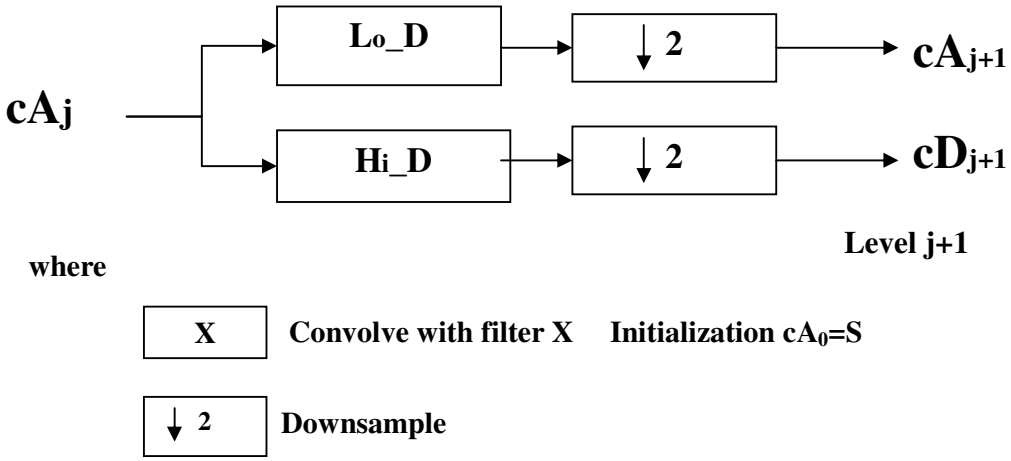


Figure2.5 Multi-Decomposition (Source: Matlab\wavelet)

So the wavelet decomposition of the signal s, analyzed at level j, has the following structure: [cAj, cDj, ..., cD1].

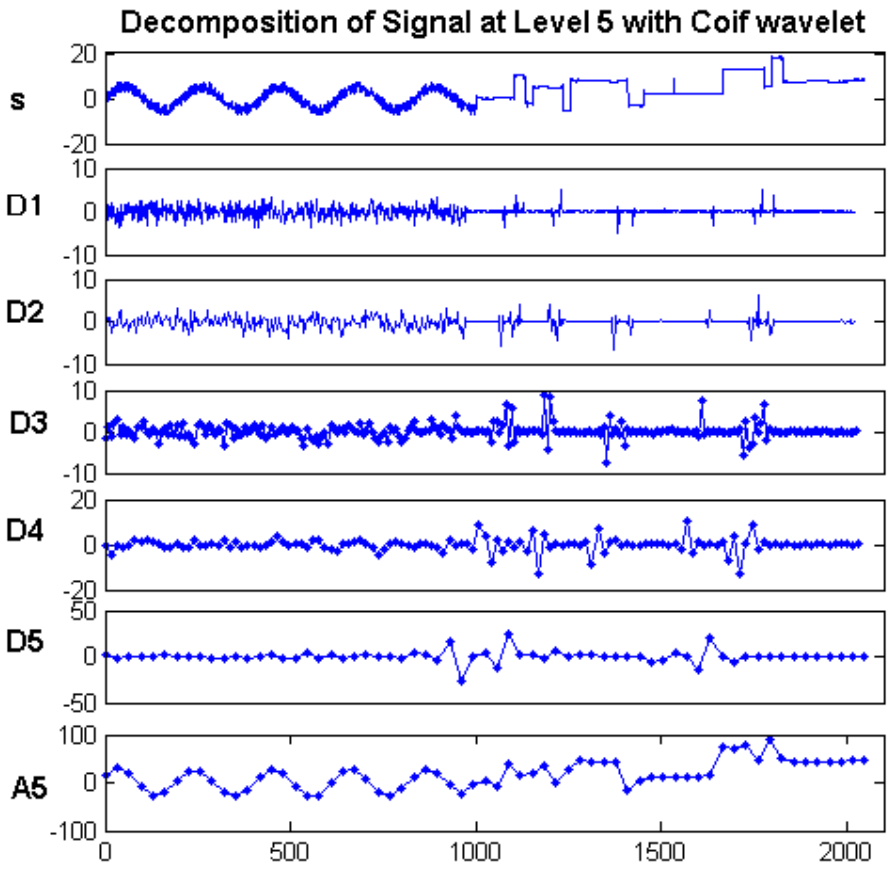


Figure2.6 Multi-Decomposition at Level 5 with Coif wavelet

The first graph in Figure 2.6 is the original signal which the front part of signal is an integrated sin signal with noise and the second part of signal is an integrate step signal with outlier. The whole original signal has 2048 points. The original signal is decomposed with Coif wavelet into five levels. D1, D2, D3, D4 and D5 are the detail signal which presents the high frequency information of original signal. A5 is the approximation signal which expresses the low frequency information of original signal. In A5 subplot of Figure 2.6, the front part of plot show a appointed sin signal. In the detail signal of Figure 2.6, the second part of plot has some vigorous variations of signal which suggest some event happens.

The other half of the story is how those components can be assembled back into the original signal without loss of information. This process is called reconstruction, or synthesis. The mathematical manipulation that effects synthesis is called the inverse discrete wavelet transform (IDWT). The reconstructed details and approximations are true constituents of the original signal. Figure 2.7 illustrates the reconstruction algorithms.

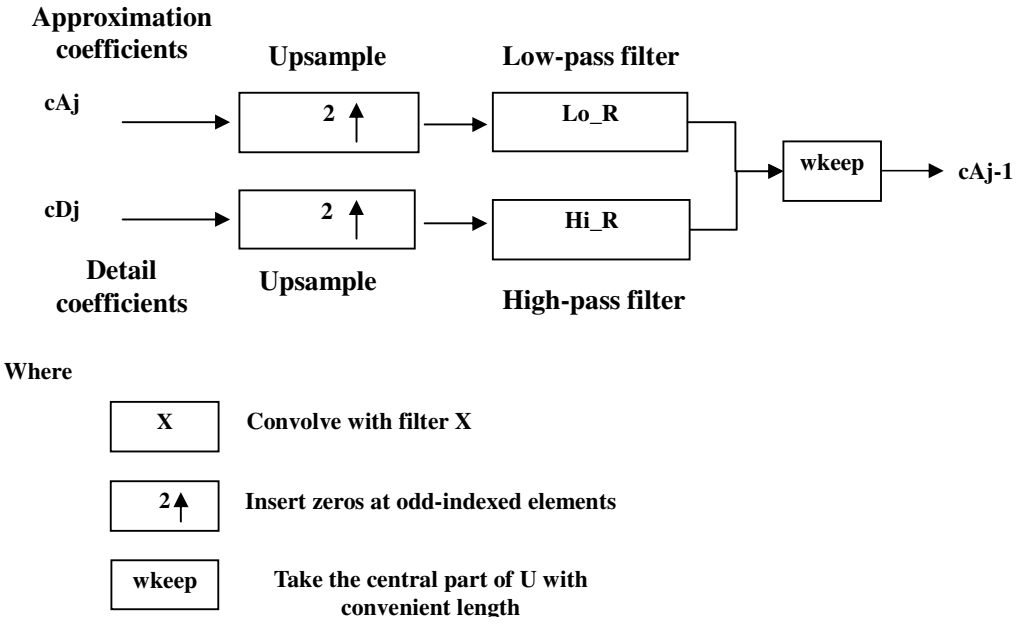


Figure 2.7 Reconstruction algorithms (source: Matlab\wavelet)

2.3.3 Evaluating Wavelet and Wavelet Transform Approach

The wavelet transform is a band pass filter with a known response function. The inverse filter can reconstruct the original time series. This filter is capable of removing amplitude regions at all frequencies. Due to this property, it can be used to remove noise, isolate a single event or multiply events. The workflow applies wavelet transform on the interpolated data to obtain the detail signal and approximated signal. Then the character of each event and noise is investigated and summarized for identification. There are many types of wavelet functions with specific properties. Choosing the most appropriate wavelet function, proper level and right wavelet transform approach are important issues.

The Discrete Wavelet Transform (DWT) algorithm has been introduced in last section. The DWT suffers a drawback in that the DWT is not a time invariant transform. Whatever its initial position is, a signal analyzed by DWT will not have the same pattern on the dyadic grid. However, detecting the right position of break point and outlier is the key step of PDG data processing. It is essential to use a kind of time invariant DWT called Discrete Stationary Wavelet Transform (DSWT) to carry out wavelet transform.

Many slightly different ways exist to handle the discrete wavelet transform. Let us recall that basic computational step in the DWT is a convolution followed by

decimation. The decimation retains even indexed elements. The decimation could be equally be carried out by choosing odd indexed elements instead of even indexed elements. This choice concerns every step of the decomposition process, so at every level we chose odd or even.

If we perform all the different possible decompositions of the original signal, for a given maximum level j we have 2j different decompositions. Let us denote by j = 1 or 0 the choice of odd or even indexed elements at step j. Every decomposition is labeled by a sequence of 0's and 1's: = 1,...,J. This transform is called the ε-decimated DWT. The basis vectors of the ε-decimated DWT can be obtained from those of the standard DWT by applying a shift and correspond to a special choice of the origin of the basis functions.

It is possible to calculate all the ε-decimated DWT for a given signal of length N, by computing the approximation and detail coefficients for every possible sequence. Of course, this is not a good way to calculate all the ε-decimated DWT, because it involves performing many computations many times. An alternative way is the stationary wavelet transform (SWT). The SWT algorithm is very simple and is close to that of the DWT. More precisely, for level 1, all the decimated DWTs (only two at this level) for a given signal can be obtained by convolving the signal with the appropriate filters as, in the DWT, case but without down sampling. Then the approximation and detail coefficients at level 1 are both of size N, which is the signal length. Figure 2.8 shows the comparison of DWT and SWT.

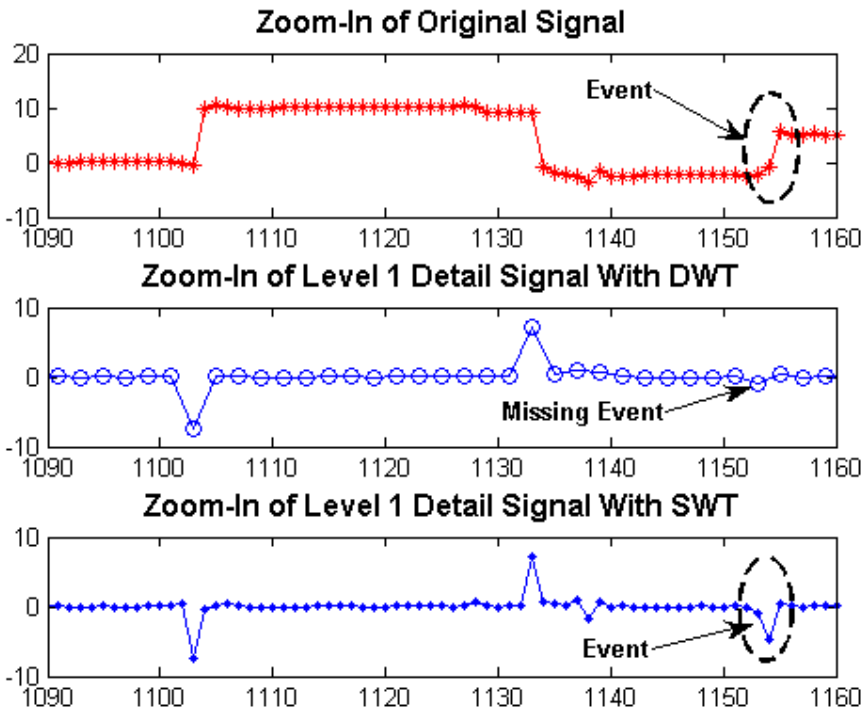


Figure 2.8 Comparisons of DWT and SWT

From figure 2.8, the detail signal with DWT is half number of data points of original signal. The detail signal with SWT has the same number of data point with original signal. The third step event happened at point 1155 in the original signal. However, we can not find in the Detail signal with DWT. That is the reason the SWT is chosen to carry out wavelet transform.

Another important issue of wavelet transform is how to choose the wavelets. There are a range of wavelets, which we can choose to perform the wavelet transform. Different wavelet has different characters. The different wavelet and level have great impact on the result of decomposition. Wavelet methods for processing permanent downhole gauges were described by Kikanni and Athichanagorn (1999 [86]). In both studies, a compactly supported nonorthogonal wavelet, namely the spline wavelet, was used for data processing and transient identification.

This study has chosen the Harr wavelet to carry out the wavelet transform because the Harr wavelet is discontinuous and thus tends to locate the break points more accurately, compared to other less compacted regular and orthogonal wavelets. The Daubechies wavelet was used in all the steps of the proposed methodology. In particular, the Daubechies 1 wavelet, corresponding to the Harr wavelet at the first level of decomposition, was used for the outlier removal. Different Daubechies wavelets at

different levels of decomposition were applied in the denoising process. The Daubechies wavelet presents best results for the synthetic and actual data studied but different decomposition levels were necessary for different flow periods.

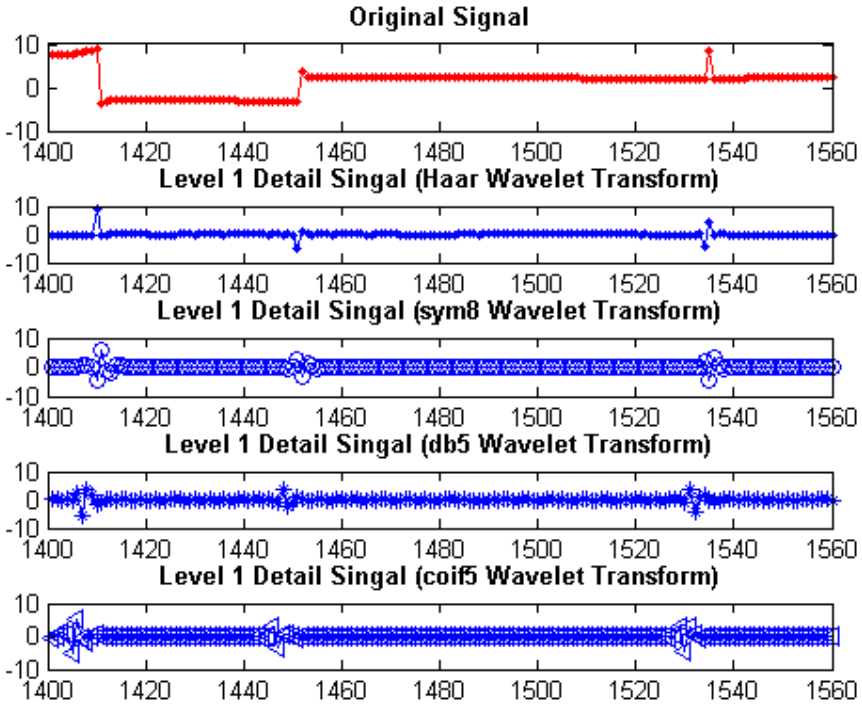


Figure 2.9 Comparisons of Detail signal with different wavelet

The Harr wavelet is explored in this section. Harr wavelets are the most compactly supported wavelets of all the orthogonal family of wavelets. When wavelet transform is applied on the interpolated data with the Harr wavelet, only level 1 data of SWT is obtained. Figure 2.8 will compare to apply different wavelet for detecting the break points. In Figure 2.8, we try four kinds of wavelet: haar, sym8, db5 and coif5 because these wavelets are all very popular wavelet. It can be found Haar wavelet can directly detect the right position of breakpoint and outlier. Other wavelet method will produce more information which can not locate the right position of break point.

2.4 Preprocessing PDG data

The preprocessing stage prepares for the following data processing. The first step in preprocessing is to tidy the original data in order to obtain familiar data. The second step is data evaluation to evaluate time intervals and data redistribution. The third step is to perform data interpolation because the wavelet transform needs signals of equal time intervals.

2.4.1 Tidying the original data

The original pressure data collected from permanent downhole gauges contain different degrees of measurement errors, such as negative values and time repetitive values. Negative values are usually observed in the PDG dataset but it is impossible to have a negative value of pressure and temperature in the reservoir. Obviously, this negative value must be caused by a gauge error. Another type of inaccurate data is repeated time, because it is impossible to have two data points at the same time. The time continuity problem may occur in some PDG systems. For example, if the data file spans multiple days then after 23:99 the time will start again from 00:00 when the date changes. These two problems can result in a wrong analysis for the dataset. Besides taking account of these two errors, some conversions are also applied to make the data easier to use. Time conversions are applied to transfer the "07/01/2007 00:00:00.125" format to hours because this is more suitable for the process and well test analysis. All pressure units are converted to the field unit (PSI). Normally, the PDG is located at the upper region of the formation. Accordingly, it is necessary to calculate the pressure in the middle of the formation so as to reflect the real information about the reservoir. The following five steps are carried out to clean the original data and convert the PDG data to the well testing analysis format:

1) Removing negative pressure values
2) Removing time repetitive values
3) Time format conversion
4) Converting pressure values from Kilo Pascal (Kpa) to psia
5) Datum shifting

Figure 2.10 shows the real PDG pressure data of B well in S-field. The S field is oil reservoir which was developed in 2006. This reservoir produced with pressure depletion because the reservoir has strong aquifer. The PDG was installed in 01/12/2006 to record pressure every 5 seconds. The sum of total data points is more than 1,300,000 in three months. The original dataset contains negative pressure values and many time repetitive values, making it difficult to understand the actual information in the PDG data.

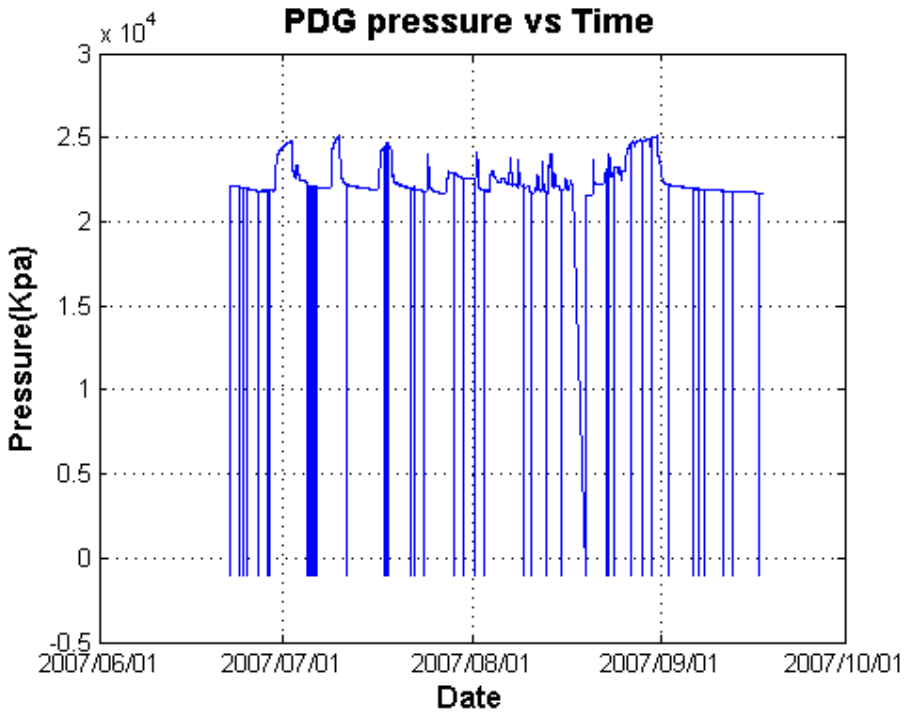


Figure 2.10 Three months S-field PDG pressure data

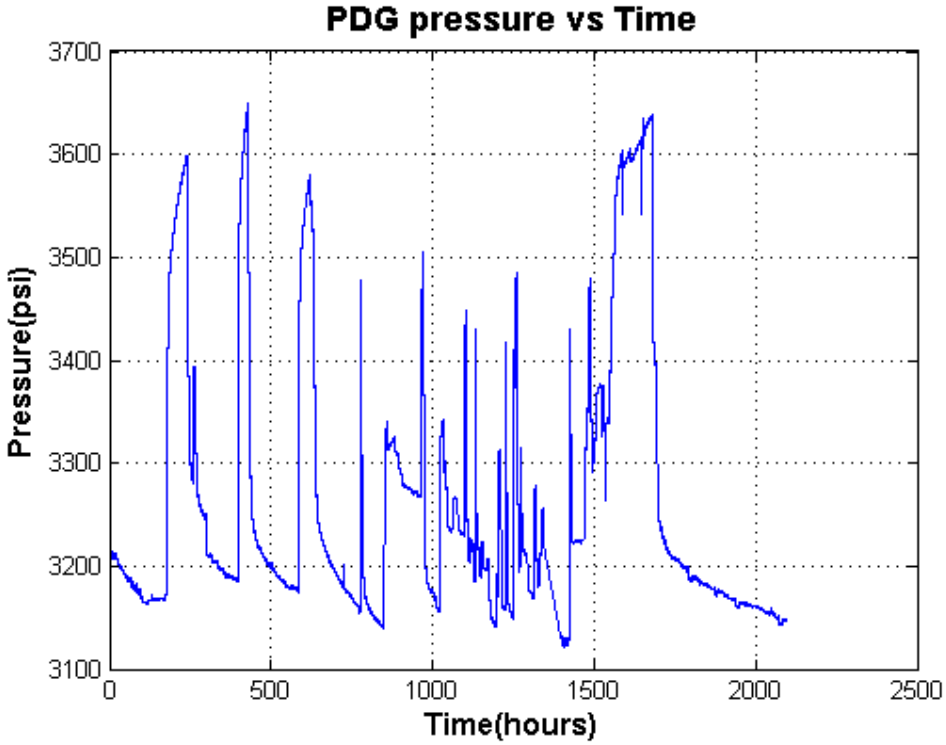


Figure 2.11 Three months S-field PDG pressure data after tidying

Figure 2.11 illustrates the clean PDG pressure data after the five-step operation. The overall trend can be found from this graph. The whole dataset consists of different BUs and DDs. There are some outlier and data gap in the whole datasets. In addition,

the time is converted from Date to hours and the pressure unit is converted from Kpa to Psi.

2.4.2 Data evaluation

The target of data evaluation is to estimate the time interval of the original PDG data. The time interval is dependent on the recording frequency which engineers set in the reservoir monitoring system. The highest recording frequency can be 1 second to record a point. However data gaps and missing data problems will often occur, causing unequal time intervals. Data gaps should therefore be marked before data processing and the average time interval should be calculated from the original dataset. Figure 2.12 shows the time intervals of a North Sea S-field PDG pressure dataset. Most of the dataset has a constant time interval but there are some gaps in the dataset. The time interval is calculated from the average of the total dataset. If the data sample comes from a normal distribution, then the sample average is also optimal. Unfortunately, outliers, data entry errors, or other glitches exist in almost all real data. The sample average is sensitive to these problems. One bad data value can move the average away from the center of the rest of the data by an arbitrarily large distance. The trimmed mean is a measure that is resistant (robust) to outliers. The idea behind the trimmed mean is to ignore a small percentage of the highest and lowest values of a sample when determining the center of sample. The average interval in Figure 2.12 is five seconds.

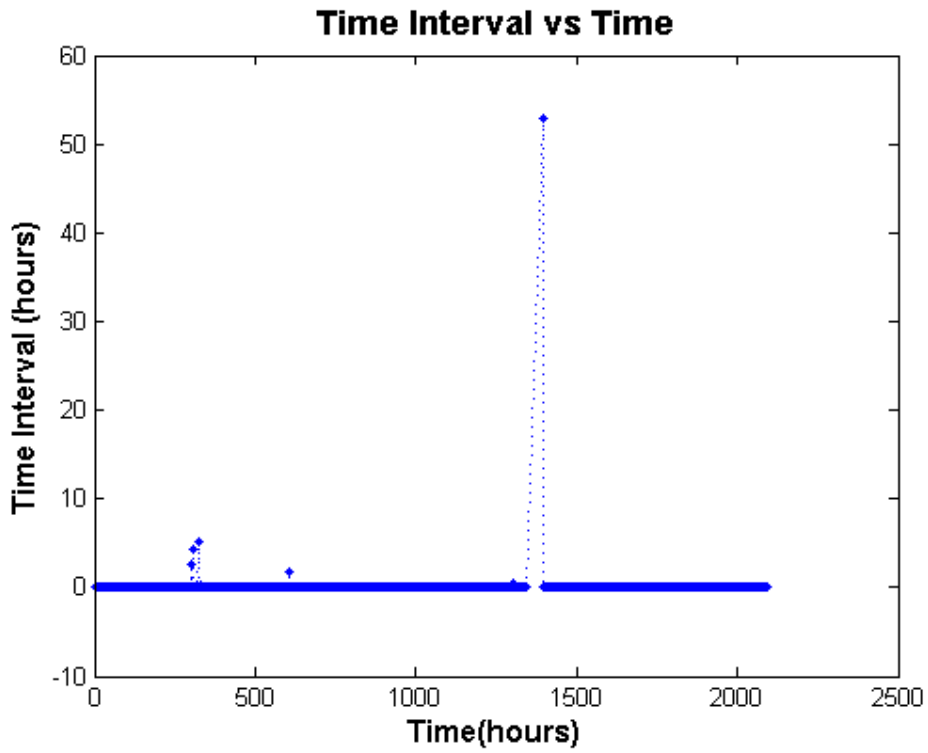


Figure 2.12 Time interval of S-field PDG pressure data

2.4.3 Interpolation

When implementing wavelet decomposition, the input dataset must have equal time intervals. In many cases, the pressure data from permanent downhole gauges are unevenly spaced, so it is necessary to interpolate the dataset to obtain an evenly sample dataset. Interpolation is the process of defining a function that takes on specified values at specified points. There are four optional interpolation methods: nearest neighbor interpolation, linear interpolation, spline interpolation and pchip interpolation. The following section will introduce the algorithm and compare these four methods.

Nearest neighbor interpolation: Nearest neighbor interpolation (also known as point sampling in some contexts) is a simple method of interpolation in 1 or more dimensions. The nearest neighbor algorithm simply selects the value of the nearest point, and does not consider the values of other neighboring points at all, yielding a piecewise-constant interpolation. The algorithm is very simple from the implementation point of view.

Linear interpolation: One of the simplest methods is linear interpolation (sometimes known as lerp). Generally, linear interpolation takes two data points, say (x_a, y_a) and (x_b, y_b), and the interpolant is given by:

$y = y_a + (x - x_a)(y_b - y_a) / (x_b - x_a)$ at the point (x, y). (2-3)

Linear interpolation is quick and easy, but it is not very precise. Another disadvantage is that the interpolant is not differentiable at the given data points.

Piecewise cubic Hemite interpolation:

Interpolation is to find values of an underlying interpolating function $P(x)$ at intermediate points. Let h_k denote the length of the kth subinterval:

$$h_k = x_{k+1} - x_k \tag{2-4}$$

Then the first divided difference, δ_k, is given by

$$\delta_k = \frac{y_{k+1} - y_k}{h_k} \tag{2-5}$$

Let d_k denote the slopee of the interplant at x_k:

$$d_k = P'(x_k) \tag{2-6}$$

Consider the following function on the interval $x_k \le x \le x_{k+1}$, expressed in terms of local variable $s = x - x_k$ and $h = h_k$:

$$P(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^3 - hs^2}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k \tag{2-7}$$

This is a cubic polynomial in s, and hence in x, that satisfies four interpolation conditions, two on function values and two on the possibly unknown derivative value:

$$P(x_k) = y_k, P(x_{k+1}) = y_{k+1}, P'(x_k) = d_k; P'(x_{k+1}) = d_{k+1} \tag{2-8}$$

If we happen to know both function values and first derivative values at a set of data points, then piecewise cubic Hemite interpolation can reproduce those data. But if we are not given the derivative values we need to define the slopes d_k somehow. Of the many possible ways to perform this, there are two methods called pchip and spline.

Shape-Preserving Piecewise Cubic (pchip) interpolation:

The pchip interpolation is a sharp-preserving which is based on Piecewise cubic Hemit interpolation. The main idea is to determine the slopes d_k so that the function values do not overshoot the data values, at least locally. If δ_k and δ_{k-1} have opposite signs or if either of them is zero, then x_k is a discrete local minimum or maximum. If δ_k and δ_{k-1} have the same sign and the two intervals have the same length then d_k is taken to be the harmonic mean of the two discrete slopes.

Cubic Spline interpolation: Spline is a form of interpolation where the interpolant is a

special type of piecewise polynomial called a spline. Spline interpolation is preferred over polynomial interpolation because the interpolation error can be made small even when using low degree polynomials for the spline. The corresponding mathematical spline must have a continuous second derivative and satisfy the same interpolation constraints. The breakpoints of a spline are also referred to as its knots. The first derivative of cubic function is defined by different formulae on either of a knot x_k and must be smooth. So, the second derivative is continuous. This means: $P''(x_k+) = P''(x_k-)$ s

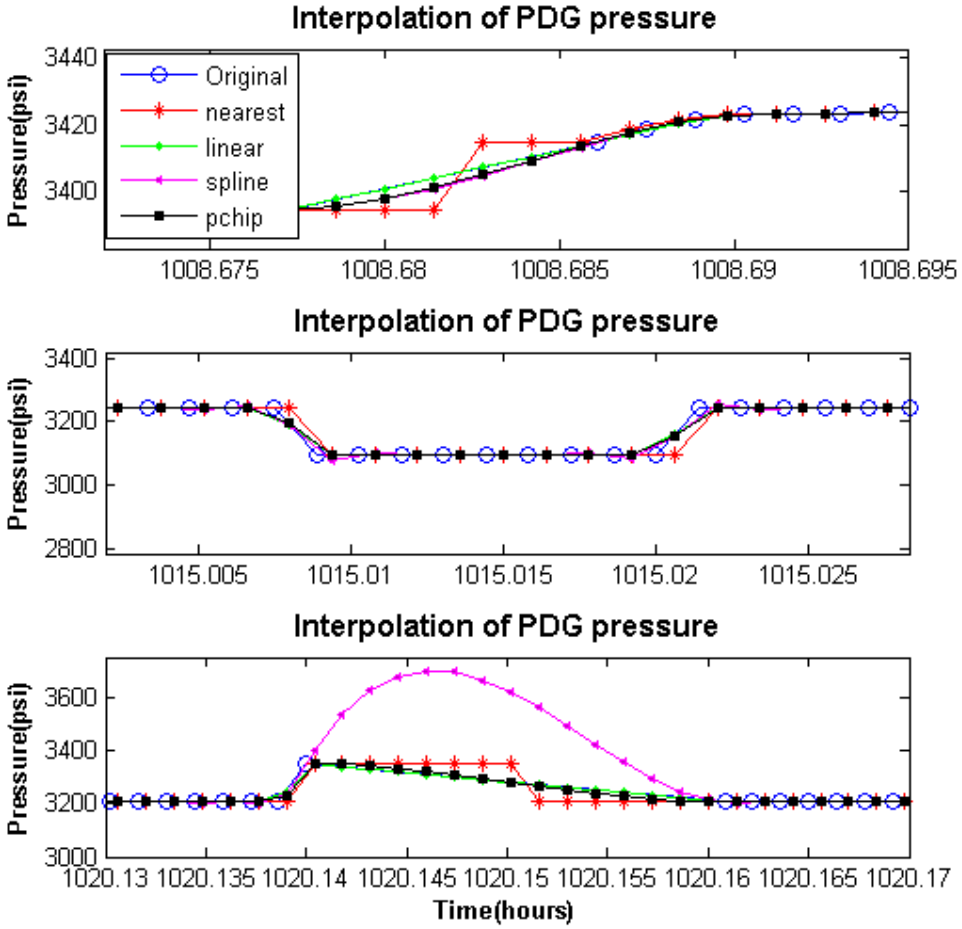


Figure 2.13 Interpolation of PDG pressure for three different situation

Figure 2.13 shows the interpolation of PDG pressure from four different approaches in different situations. There are three synthetic PDG pressure data from oil reservoir. In first plot, there is a gap in the beginning of the transient part. The nearest approach produces a break point in the gap. The linear approach represents this part with a straight line. However, the spline and pchip will give more reasonable interpolation.

The second plot shows a step outlier. The pchip gives a better result than the spline method. In plot 3 there is an outlier and a gap which is located near the outlier. Plot 3 shows that the interpolation can introduce more outlier data points into the dataset. Athichanagorn mentioned this problem in his thesis. That is the reason why he suggests performing outlier removal without interpolation. In our approach, we confirm that we perform interpolation before wavelet decomposition because this is the theoretical basis for wavelet.

2.5 Outlier removal

Real pressure data are also subject to possible fluctuation: for example an unusually large disturbance, or temporary sensor or transmitter failure. It is important that such outliers are not allowed to affect the models too strongly so they are therefore removed from original data. The outlier can be categorized as actually being two types of outlier: single spike outliers and step outliers. Figure 2.14 illustrates the the normal spike outlier and step outlier.

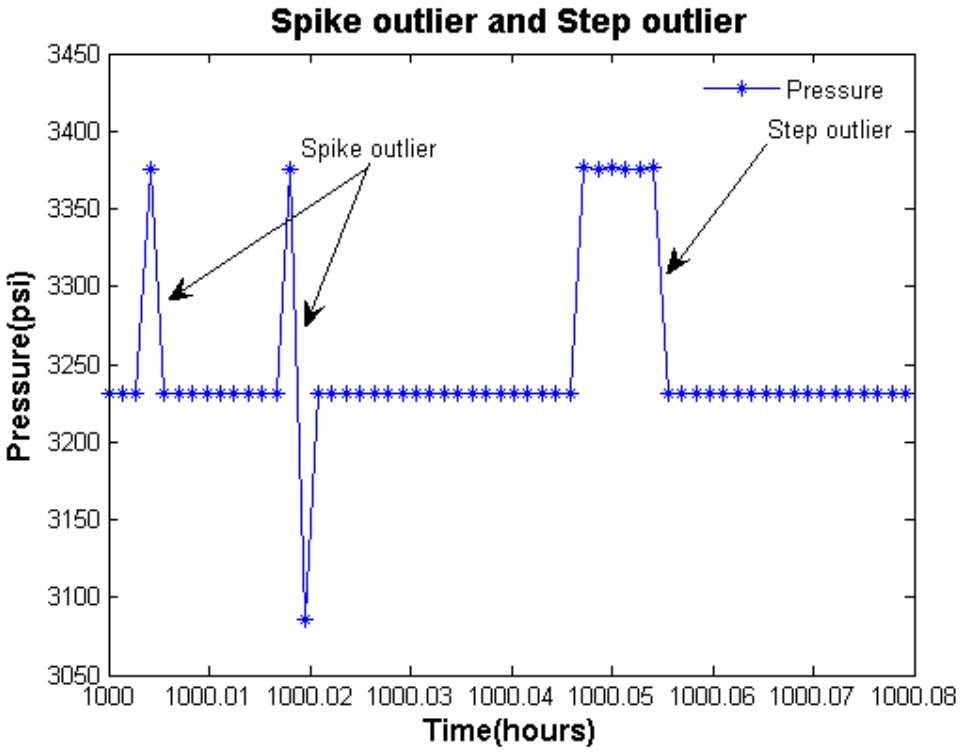


Figure 2.14 Spike outlier and Step outlier

Athichanagorn first presents an approach to remove outlier with a wavelet method. However, his approach can only remove the single spike outlier and he also did not give a flexible method of setting the threshold. The first improvement was made by Khong.

He found another type of outlier which was named the step outlier. This step outlier cannot be removed by the Athichanagorn method. Khong suggests removing this step outlier manually. The other way to remove the outlier is based on the definition from Matlab. Olsen's (2006 [101]) investigations show that a median filter will work better. The method works such that every point that has larger deviation from the trend than the estimated noise level is defined as an outlier. This method needs to be given a specific window size. A different window size leads to a completely different result. In addition, this method can not remove the step outlier.

The current methods have limitations when processing the step outlier. The single spike outliers can be removed with current methods when occurring in the stable pressure part. Wavelet methods have a problem when trying to remove single spike outliers which occur in the transient part and no approach can remove the step outliers. This part will introduce a new algorithm to remove outlier and then illustrate some examples for new algorithm.

2.5.1 Algorithm of outlier removal

Outlier removal is mainly dependent on the definition of the outlier. The more an outlier is studied, the more information can be identified. There are two types of outlier that can be identified: spike outlier and step outlier. The spike outlier is one data or two data away from the local tendency of the dataset. The spike outlier can appear in both the stable part of the dataset and in the transient part of the dataset. The step outlier is a part of the dataset away from the local tendency of the dataset. it usually occurs in the stable part of the dataset. The step outlier can be caused by mechanical problems in the gauge. It also is induced from the interpolation when the time interval is greater than the interpolation time interval.

In this research, a new algorithm is developed from the wavelet method and windows approach to remove these two types of outlier. As we know, the wavelet method mainly processes the PDG data in frequency domain, while the windows approach is used in the time domain. The new approach will integrate the information in both domains in order to cross check the information. The algorithm is separated into three steps to remove the outlier: grouping the detail signals, identifying the outlier and removing the outlier.

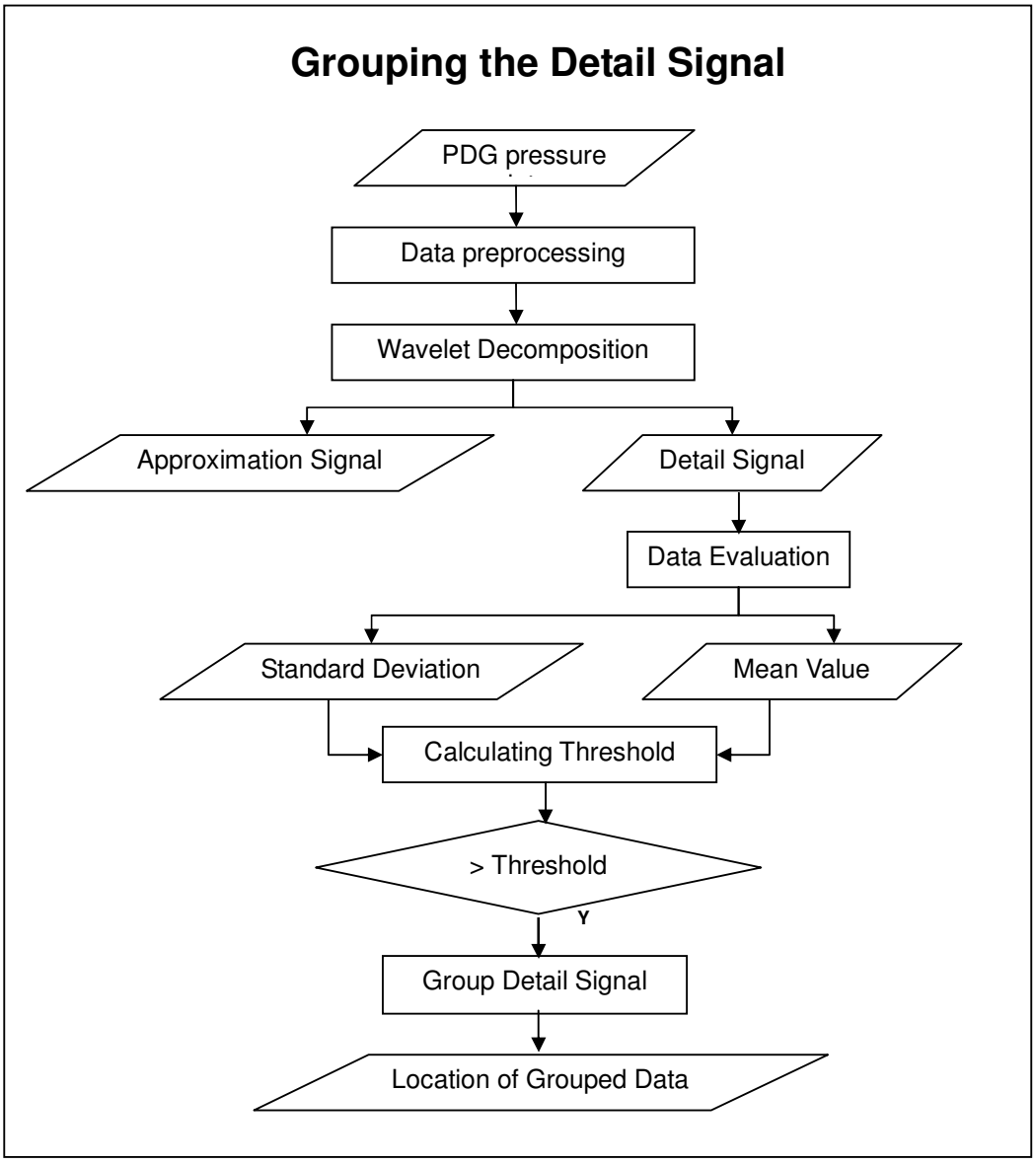


Figure 2.15 the flow chart of grouping data for outlier removal

Figure 2.15 illustrates the first step of outlier removal algorithm. After data preprocessing, the PDG pressure have been cleaned and interpolated into equal time step PDG pressure. This equal time step PDG pressure is then transformed with harr wavelet. The PDG pressure is decomposed into two kind signals: Level 1 detail signal and Level 1 approximation signal. We main focuses on the detail signal because the characters of outlier are presented in the detail signal. However, we do not interest all data information in the detail signal because the detail signal contains lots of noise information except the outlier information. In order to identify the singularities in the detail signal are caused by the outliers, we use one threshold to select all the singularities whose amplitude in detail signal is bigger than the amplitude of noise in

detail signal. Data evaluation is performed to calculate the mean value and standard deviation of detail signal. The mean value of dataset is from the trimmed mean and the deviation is calculated from InterQuartile Range (IQR). The IQR is a robust estimate of the spread of the data, since changes in the upper and lower 25% of the data do not affect it. If there are outliers in the data, then the IQR is more representative than the standard deviation as an estimate of the spread of the body of the data. The data evaluation calculates the mean value μ and standard deviation σ of the detail level 1 values. After calculate the mean value and standard deviation, the upper and down threshold value is calculated with the form $\mu + \sigma$ and $\mu - \sigma$. This threshold value should be bigger than the amplitude of all noise signals in the detail signal and smaller than the amplitude of outlier signals. All positions are marked where the amplitude of detail signal is bigger than the threshold. Then all marked data positions when the time interval is not more than three times the interpolation step are grouped.

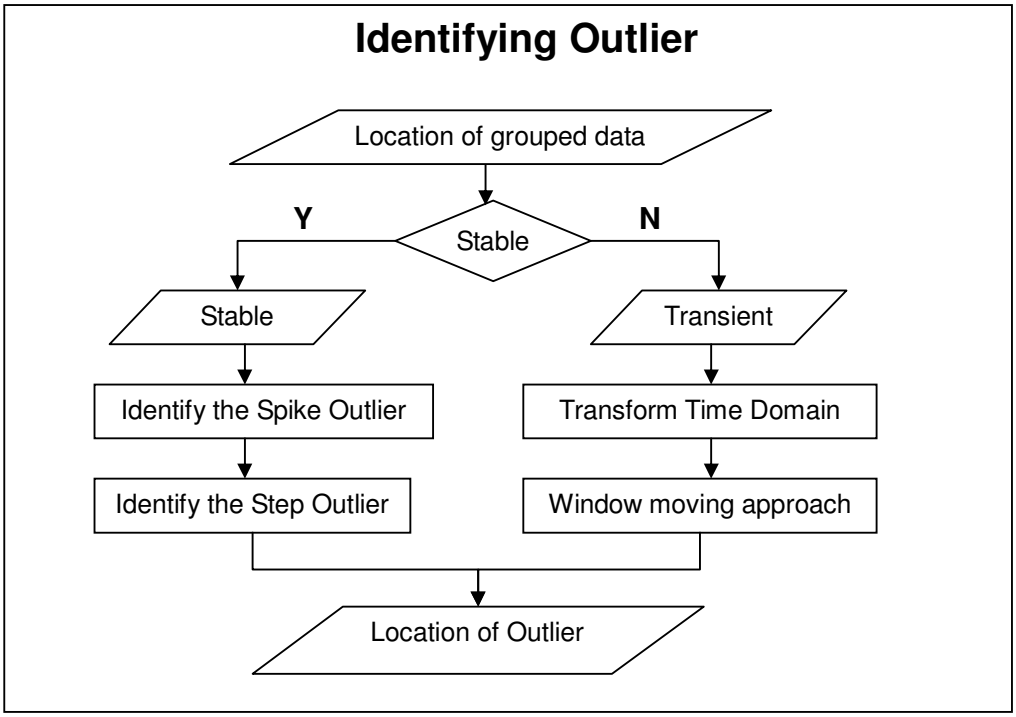


Figure 2.16 Flow chart of identifying the outlier

Figure 2.16 show the flow chart of identifying the location of outlier. The second step is to identify the location of spike outlier and step outliers. The algorithm compares the true pressure value on both side of every selected group. If the value is smaller than the normal noise value of PDG pressure, this group is located in the stable dataset. If the value is bigger than the noise value of PDG pressure, this group is in the transient part.

If the grouped data is in the stable dataset, the algorithm will detect the spike outlier first. The groups are selected when the size of group is equal to 2 and 3. The location of data in group will be recorded when the detail signal is opposite. The second step is to detect the step outlier. The step outlier is detected from the two opposite signals on either side of the step outlier, in high frequency. The algorithm seeks all the nearest two groups in which the size is one and the two values are opposite. The outlier is difficult to be identified when it occurs in the transient part. Our algorithm uses the windows method to identify this type outlier in time domain. The windows method gives a flexible window size according to the size of group. If the value is four time standard deviation away from tendency, the point will be considered as outlier. The location of outlier will be recorded in the algorithm.

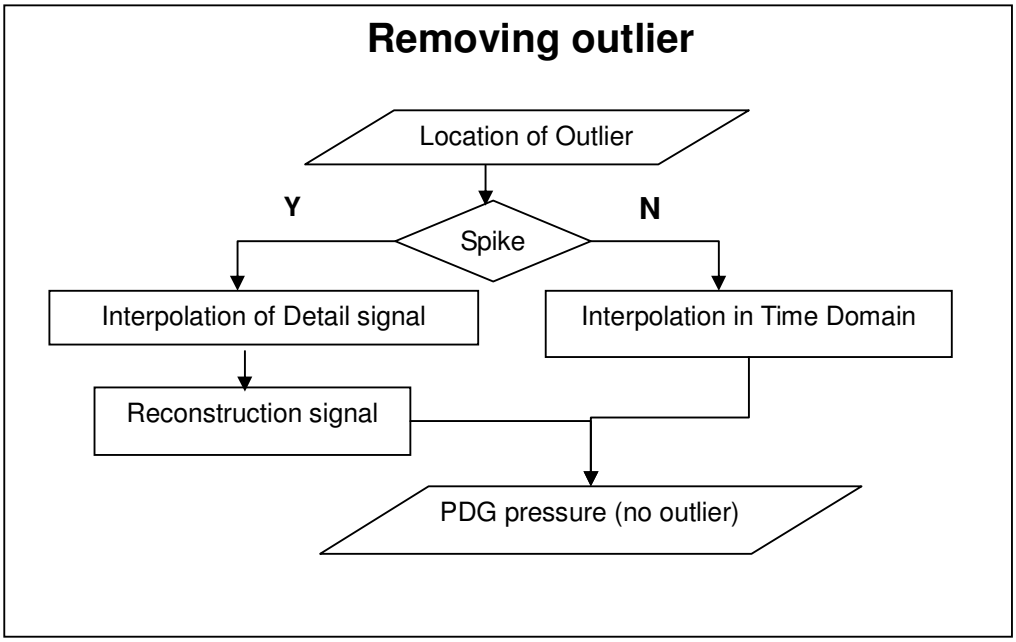


Figure 2.17 Flow chart of outlier removal

Figure 2.17 illustrates the flow chart of outlier removal. This step is to remove the outlier from the original signal. There are two ways to remove the outlier. One way is to interpolate the high frequency signal value at the location of spike outlier. The inverse wavelet transform is used to reconstruct the original signal. The second way is to interpolate the original pressure data in time domain when the step outlier data is located in the dataset.

2.5.2 Field example of outlier removal

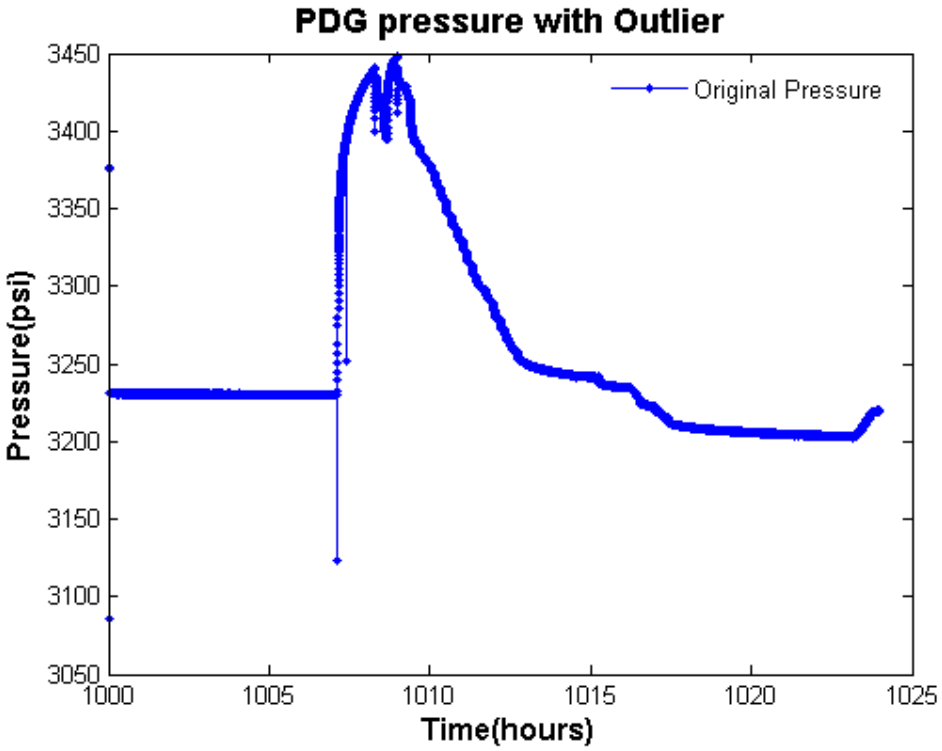


Figure 2.18 Part of PDG pressure data with outlier

Figure 2.18 shows a real part of PDG pressure data with spike and step outliers. This data is from north-sea M field. The high-quality Upper Jurassic turbidities reservoir lies within a stratigraphic trap. The reservoir has a low gas / oil ratio and will require pressure maintenance through water injection from the onset of production. The reservoir also features excellent rock quality with up to 360ft of high porosity, high permeability sands with an oil saturation of up to 94%. The oil consists of medium sour crude (32.6° API, 1.4% sulphur) with minor wax and asphaltenes.

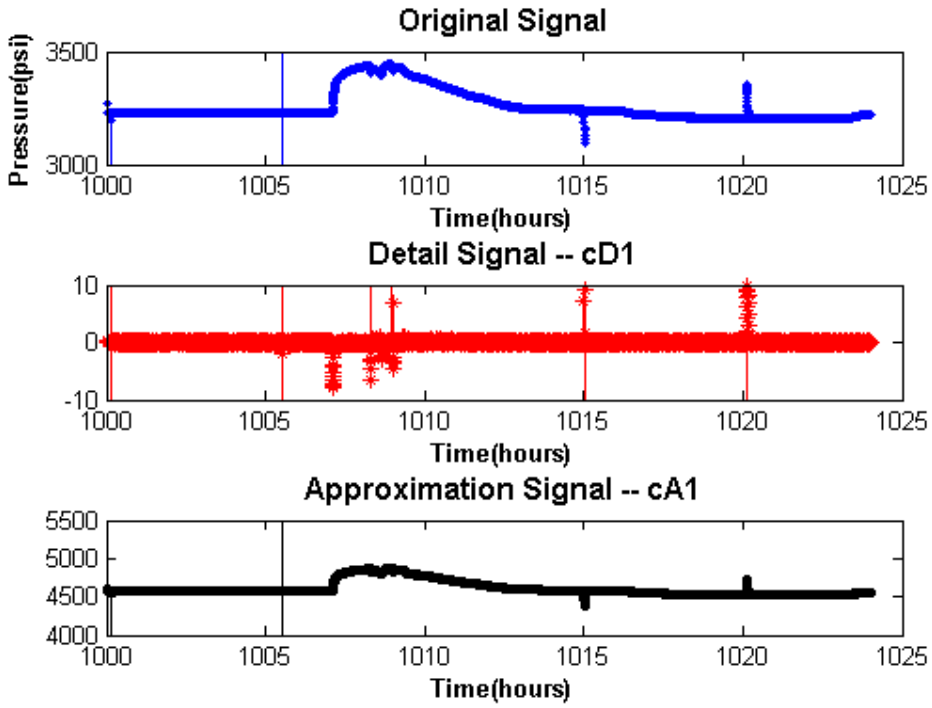


Figure 2.19 Decomposition for PDG data outlier removal

Figure 2.19 shows the detail signal and approximation signal after wavelet transform. We can found the outlier can cause some fluctuation in the detail signal.

After the wavelet decomposition, Figure 2.20 illustrates how to group the detail signal. The green line in figure is the PDG pressure with outlier. We can find some outlier at 1007.15hours and 1007.4hours. The blue line is the detail signal after the wavelet transform. The sharp fluctuation at 1007.15hours and 1007.4hours shows some events happen here. So, we need detect this fluctuation with the threshold which is red line in the figure. The threshold can calculated from data evaluation of detail signal. Then we select all data point if the amplitude of detail signal is bigger than the threshold. If the data points are neighborhood data points, these data points will be treated as the individual group data. After grouping data, we can identify the spike outlier and step outlier in figure 2.21.

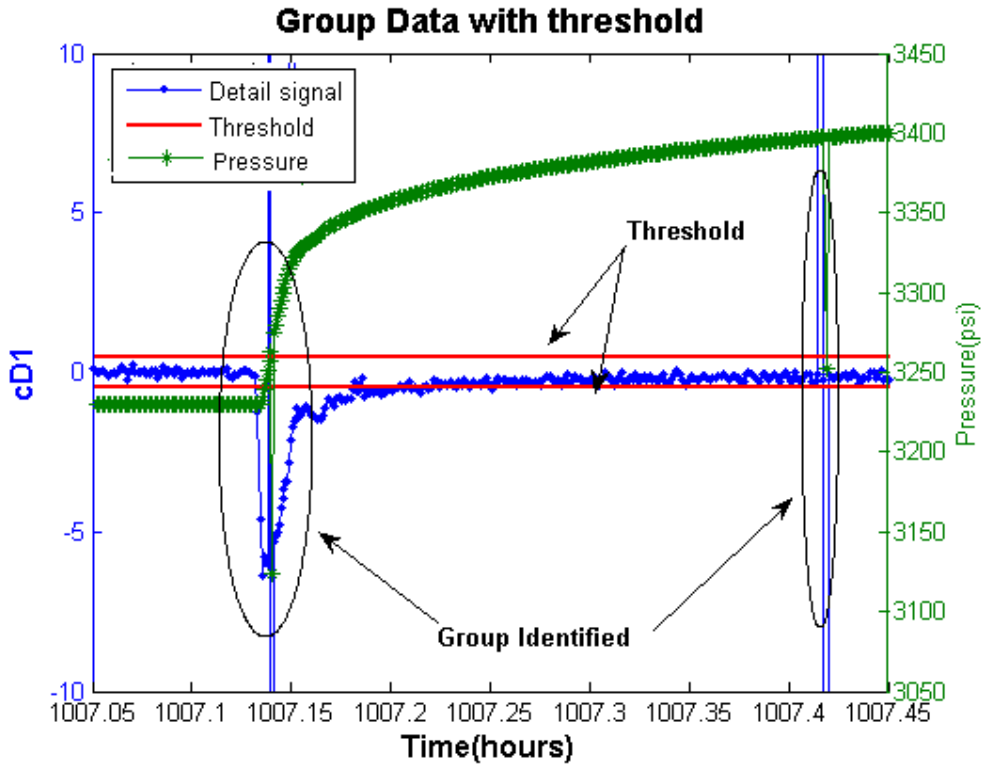


Figure 2.20 Zoom-In of Grouping data for PDG data outlier removal

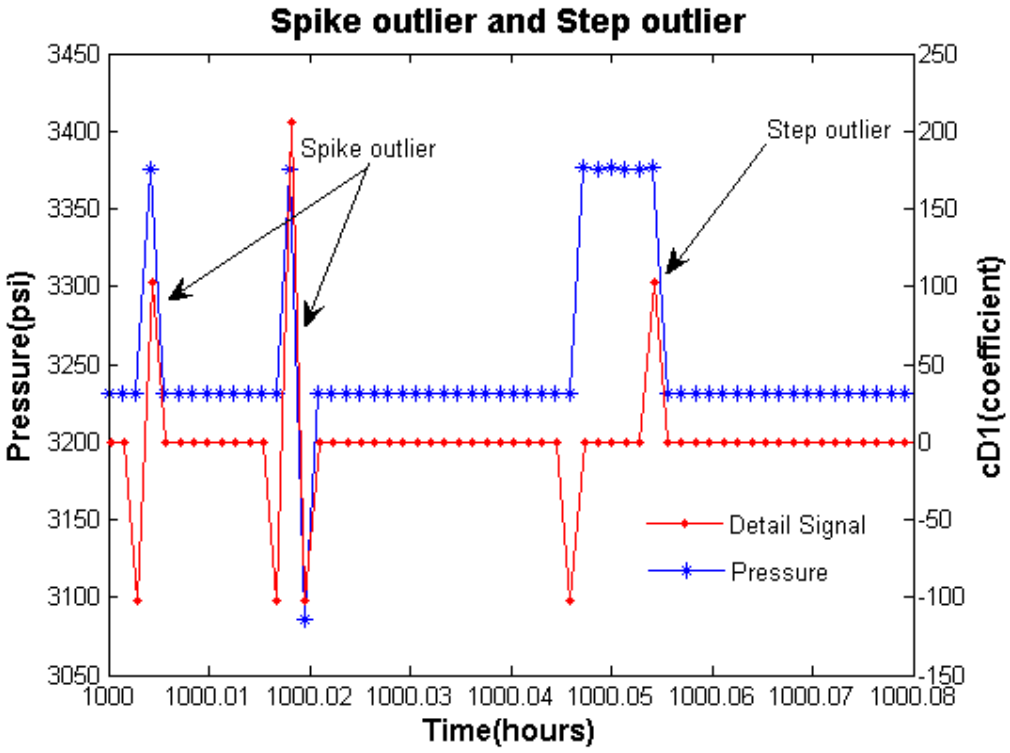


Figure 2.21 Identification of spike outlier and step outlier

After the identification of the location of outlier, the location of spike outlier is reconstructed with interpolated detail signal and approximation signal. The step

outlier is removed and the new data will be filled with interpolation of data in time domain. Figure 2.22 shows part of outlier removal pressure.

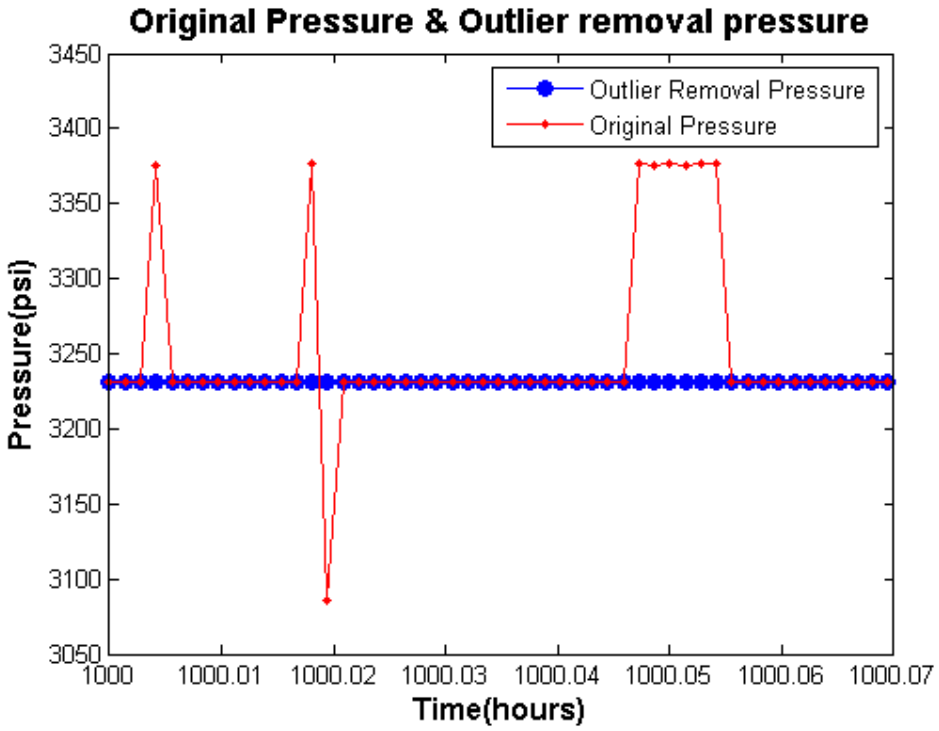


Figure 2.22 Zoom-In of PDG pressure after removal outlier

2.6 Flow event identification

The transient identification is a methodology aimed at automatically identifying the time at which the pressure trend changes as a consequence of a rate change. This step is the key to carrying out data processing for PDG pressure data. It also the most difficult part of data processing. The critical issue in transient identification is to select the proper break point of every flow period. Since 1999, more than twenty papers and thesis have investigated how to perform transient identification. Initially, wavelet transforms and the use of wavelet modulus maxima have been used for singularity and transient detection (Athichanagorn 1999[106], Khong 2001[53], Ouyang 2002[60], Nomura 2006[74]). The second direction followed is pattern recognition (Thomas 2002[85], Olsen 2005[102]). Other approaches have been the slope method (Viberti 2005 [36]) and the SG-Filter (Rai 2007[89]).

Athichanagorn's method gives the basemis for the wavelet method. Rai presents a considerable improvement which introduces the Harr wavelet and SWT into the algorithm. But the limitation of wavelet methods is that only the medial level flow

period can be detected. The smaller flow periods will be missed. In addition, wavelet modulus maxima can only identify the beginning of a flow period without any fluctuation. Rai's SG-Filter approach gives a way to smooth the original data and he tried to detect the flow period with the derivative of the data. This approach has limitations in processing abnormal wellbore storage phenomena and in identifying the right breakpoints. Pattern recognition is to identify the transient flow period with a predefined pattern. This approach will give wrong result if the predefined pattern does not match the real case and is very sensitive to the outlier and noise. Viberti's slope method defines $\lambda_{buildup}$ and , λ_{DD} which depend on the user's observation, and decides the break point when the slope of data point is bigger or smaller than the slope threshold. This method cannot identify the right break point because user can not give a constant $\lambda_{buildup}$ and λ_{DD} in one dataset.

2.6.1 Algorithm of flow event identification

The new algorithm is based on the wavelet method but it is not the wavelet modulus maxima approach. This approach is more robust than the modulus maxima method. The new algorithm can not only identify the major flow periods but also detect small flow periods. This algorithm also considers the abnormal response of wellbore storage. The speed of the algorithm is very fast. The algorithm can be summarized into three steps: grouping the detail signal; identifying the flow event in the high frequency signal; refining break point.

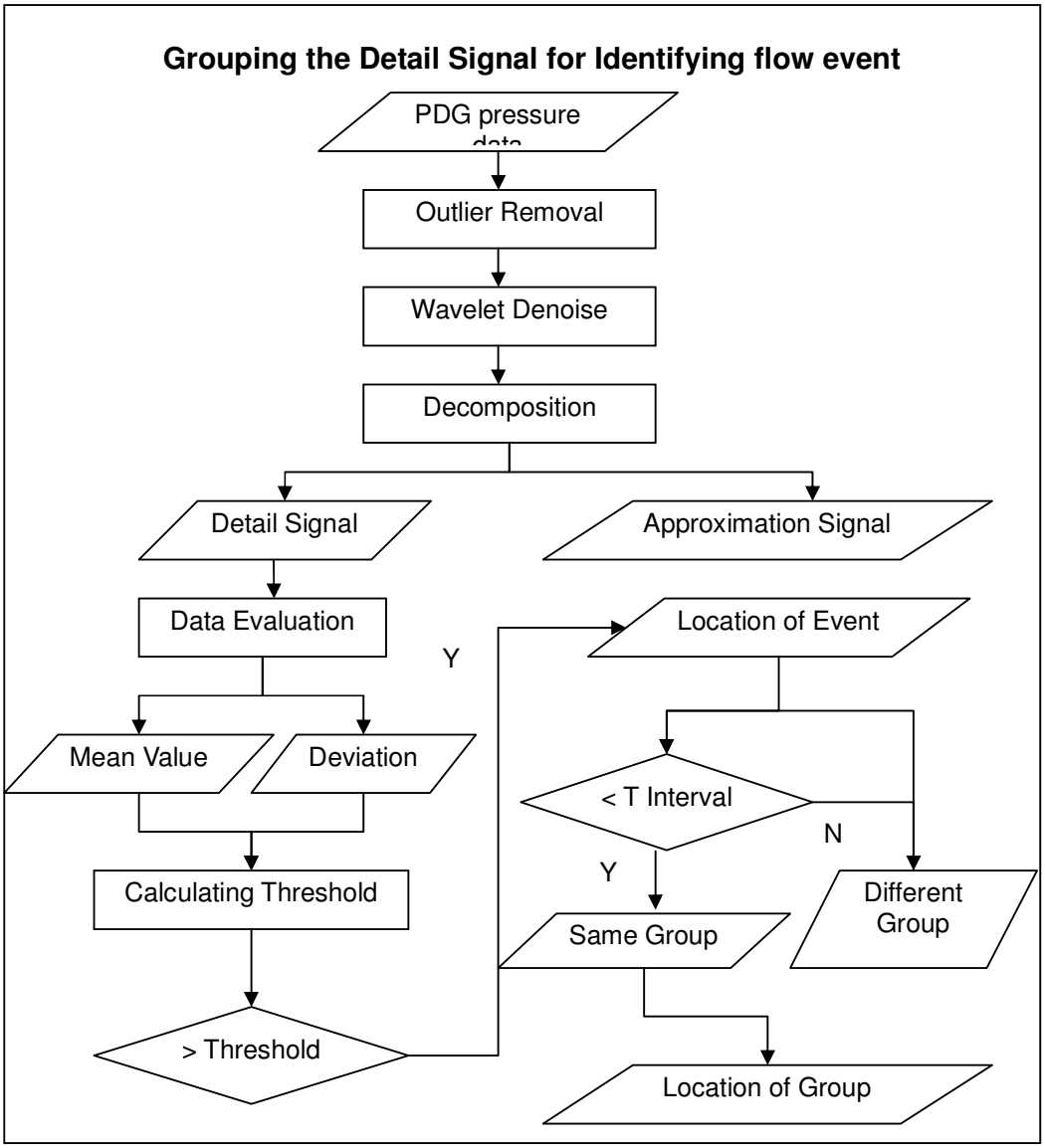


Figure 2.23 grouping the detail signal for identifying the flow event

Figure 2.23 illustrated the flow chart of grouping the detail signal for identifying the flow event. After the outlier removal, the Gauge event has been filtered. The next step is to detect the flow event in order to divide the original PDG pressure into separated flow period. The next step is to denoise the outlier removal PDG pressure. Denoising is a procedure that is applied to the data to reduce scattering and fluctuations in the data values in order to extract the representative features from the data. We apply the wavelet method to denoise. To reduce the noise level, the wavelet detail signals with magnitudes smaller than a certain threshold are set to zero and the denoise signal is constructed using the smoother detail signals. The detail algorithm will be discussed in section 2.8. After denoising, the clean PDG pressure is decomposed into detail signal and approximation signal.

From the observation and experience of processing PDG data, we found there is likely to be some violent variation when the flow event happened. This violence shows as fluctuation for a short period in the detail signal after the wavelet decomposition. These fluctuations are normally greater than noise deviation after data denoising. The threshold is set to locate the position of these fluctuations. The first step is to perform data evaluation for the detail signal to find mean value and standard deviation in order to decide the threshold. The data evaluation calculates the mean μ and standard deviation σ of the detail level 1 values. This threshold value is first set at $\mu+10*\sigma$ or $\mu-10*\sigma$. But this threshold may be too high so that small flow events can not be detected. This high threshold divides the whole part into different small parts. For every small part, the algorithm will try to select a smaller threshold to group data. This step is repeated until we can obtain the stable number groups.

There are still some other conditions, which should be considered. The first condition is the minimal flow interval time, which is the minimal duration between the neighbouring flow periods. From our experience, this parameter is assigned as 2 minutes because 120 seconds is small enough for two flow events. The 2 minutes flow period is the maximum resolution of this algorithm. The second condition is the number of data points in one group, which must be more than three points because the number of wellbore storage data point is normally more than 3 points.

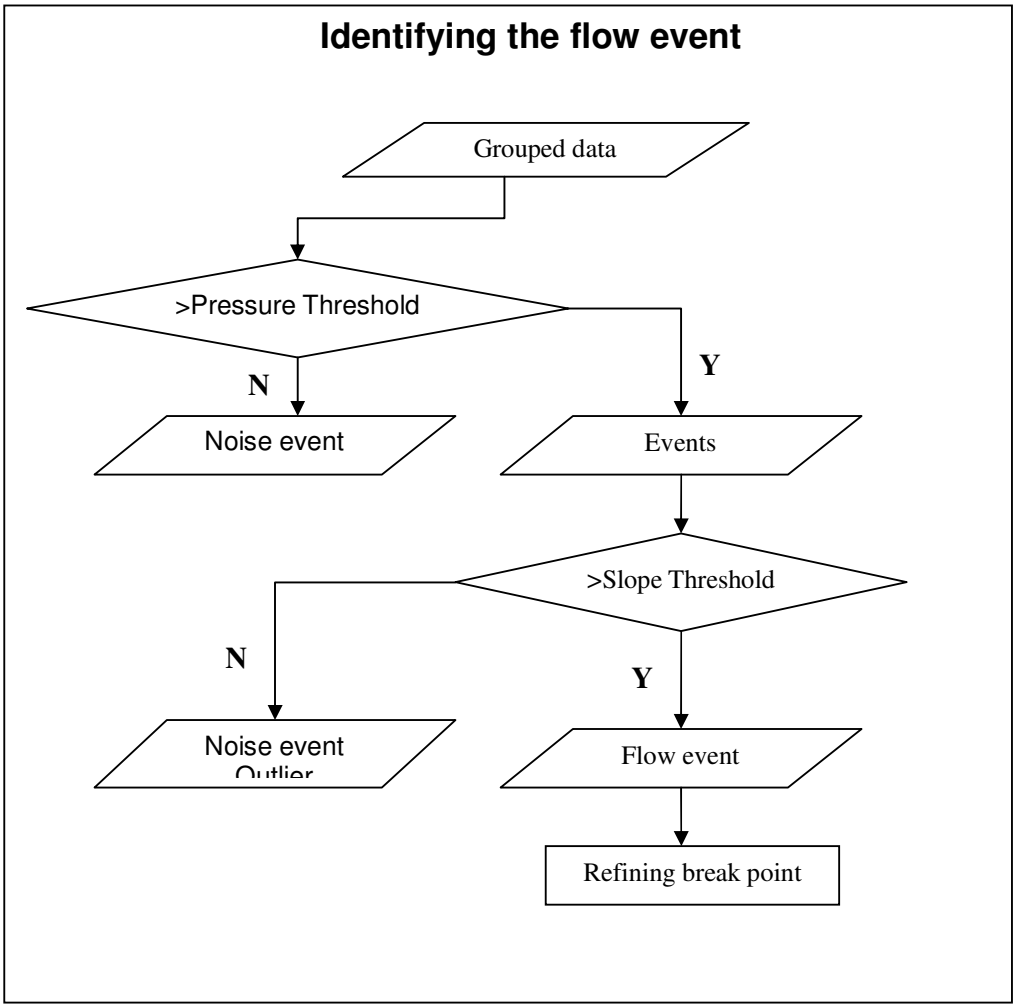


Figure 2.24 Flow chart of identifying the flow event

The grouped event is not only flow event but also some noise event. Therefore, the next step is to identify the right event group for the flow period. For this, it is essential to filter the noise events. There are several conditions which the flow events should satisfy. The first condition is found by inspecting the original pressure gap between the several points on both sides of grouped events. This pressure gap may be larger than a pressure threshold which can be estimated from the original data. The data evaluation calculates the mean μ and standard deviation σ of the data gap between the assigned window sizes. This default threshold pressure value should be $\mu+\sigma$ or $\mu-\sigma$. If the grouped event data cannot meet this condition, this grouped event is the noise event. This always filters any noise event which occurs in the stable part of flow period. However, this condition can not remove a noisy flow event at the beginning of the flow period. The second condition is to compare the slope on the both sides of the grouped data. If the slope of both sides is the same, this group of data is considered as a noise event. If the slope of both sides is different, this grouped data is flow event.

The previous step gives only an approximate region of floe event. However, the right break point is critical for the subsequent pressure analysis. As we observed in the PDG data, the first point of the flow period will be a sharp change when the well production rate changes. This first point can be considered as the outlier compared with the stable data at the end of forward flow period. Our algorithm applies a moving window method to identify this outlier. The algorithm first gives a flexible window size which depends on the size of front flow period. The last point of the window is the first point of the grouped data, and this window will move from the first point of group to the end of the group. If the windows find at least three outliers, then the first outlier will be considered as the break point of this flow period.

2.6.2 Field examples of flow event identification

The transient identification algorithm has been applied to different types of real data to test logic and performance of algorithm. The following section will introduce the tested examples.

I. Field Example 1 – Benchmark example:

Figure 2.25 shows the PDG pressure data and denoised PDG pressure after outlier removal. The dataset is the same data which was tested to carry out the outlier removal. In this case, we denoise the noisy signal using a soft heuristic threshold and scaled noise option, on detail coefficients obtained from the decomposition of the original signal, at level 5 by sym8 wavelet. Figure 2.26 shows zoom-in of the denoised signal and original signal. When the figure is zoomed in, we find the wavelet denoise is smooth in the stable part of the pressure signal but it causes a fluctuation at the end of stable part and the beginning of flow period. So, wavelet denoising can compress the fluctuation in the stable period and magnify the signal at the beginning and end of flow period. This means that the first points identified are always not the right break point of flow event.

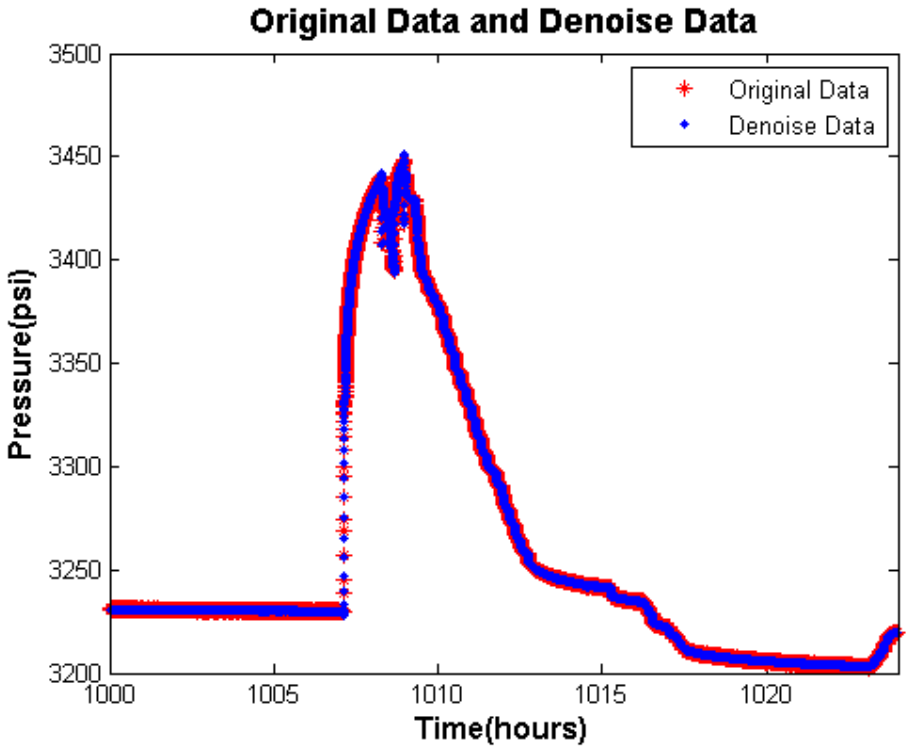


Figure 2.25 Original data and denoise data of Examples 1

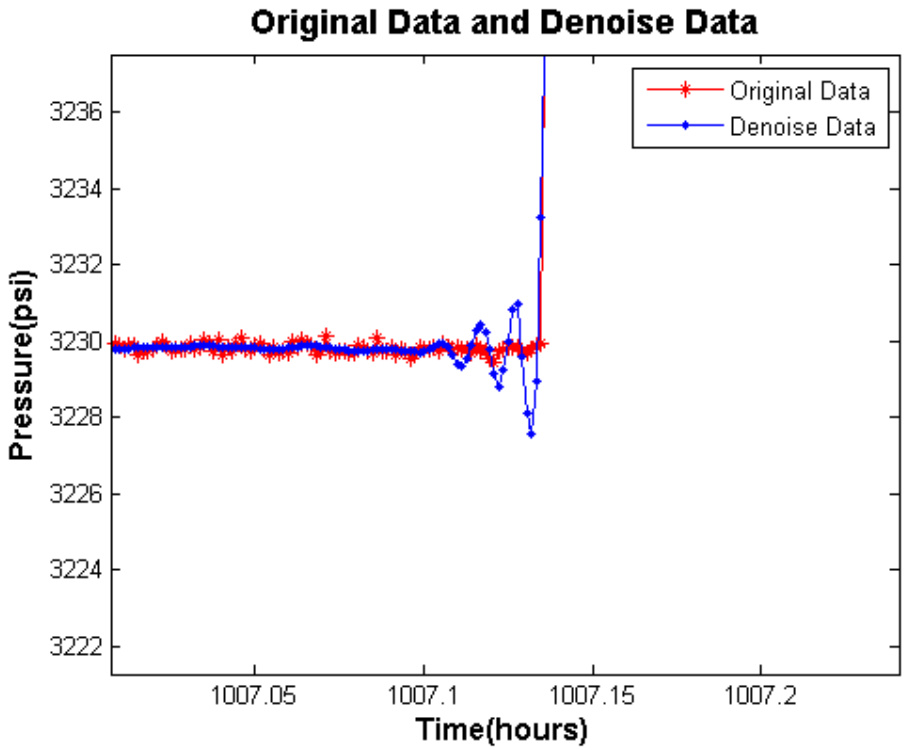


Figure 2.26 Zoom-In of data denoise of Examples 1

After denoise data, the normal wavelet modulus maxima approach decomposes the signal into certain levels. Then a slope threshold is applied to find the local maximum

value in the medial level detail signals. The problem of the modulus maxima approach is that there is no local maximum value to be found in the detail signal.

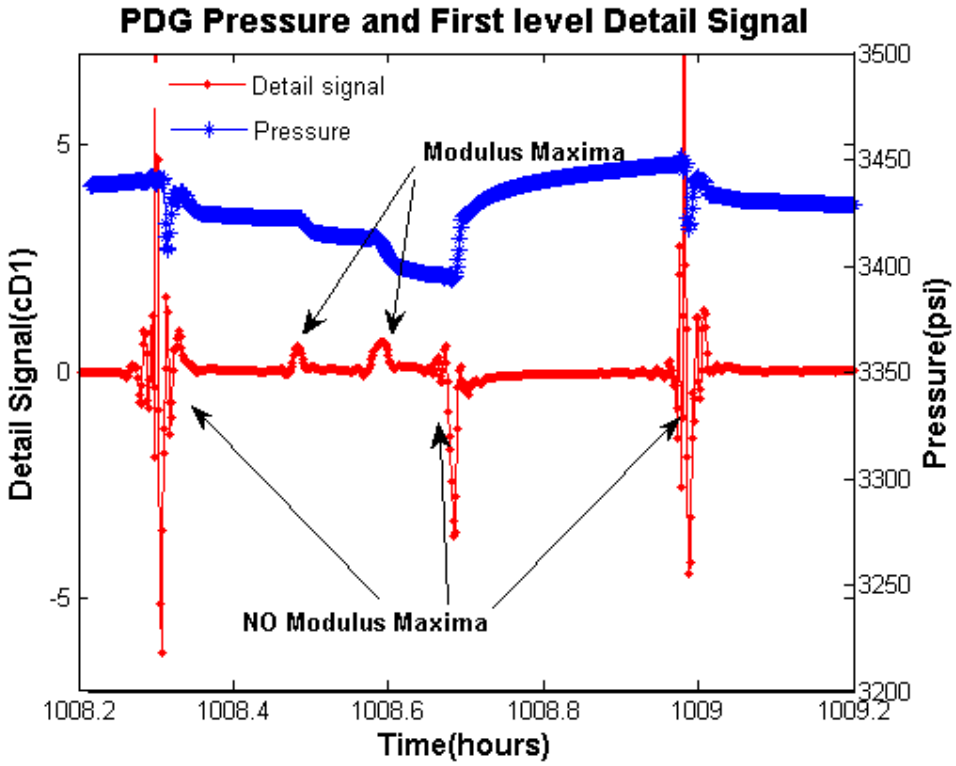


Figure 2.27 Zoom-In of first level detail signal and original pressure signal of Example 1

Figure 2.27 shows five flow events. The second and third event can be detected with the modulus maxima approach but it is very difficult to identify the first, fourth and the last flow events because there are no local maxima of data. Thus the modulus maxima are not suitable to identify the transient event in this type of situation.

The other problem of the other wavelet approach is that some small flow periods will be lost when the medial signal is considered. Figure 2.28 shows these phenomena. Therefore our algorithm simply decomposes the signal into the first level. All large and small flow events are illustrated in the first level detail signal. However, the first level information sometimes may be flooded by noisy information. That is the reason why the original data should first be denoised.

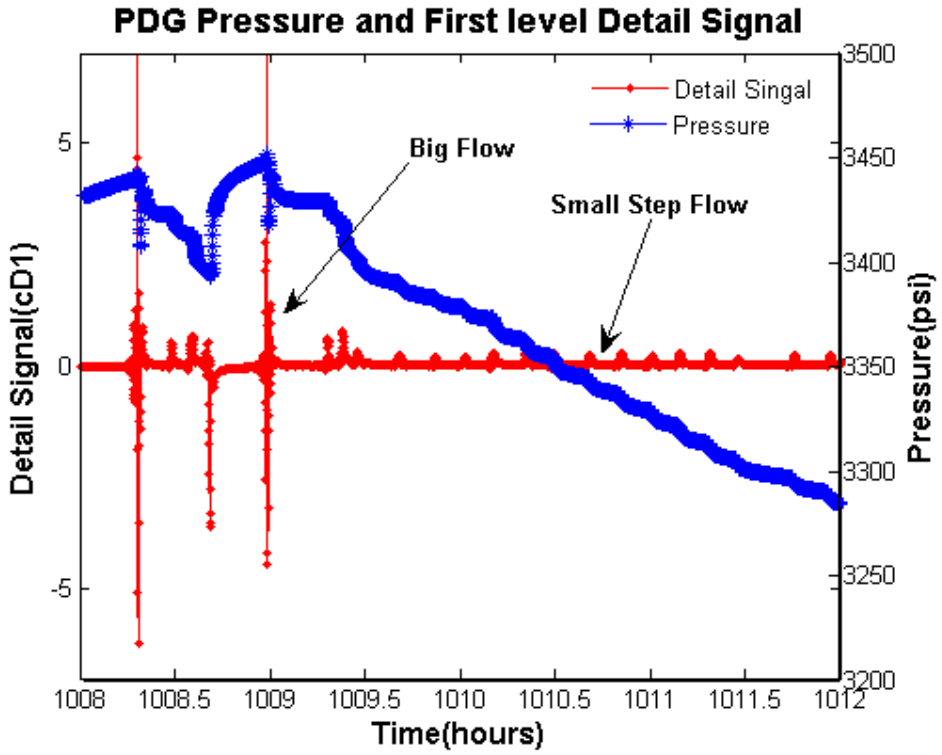


Figure 2.28 Large rate variation and small rate variation of flow event of Example 1

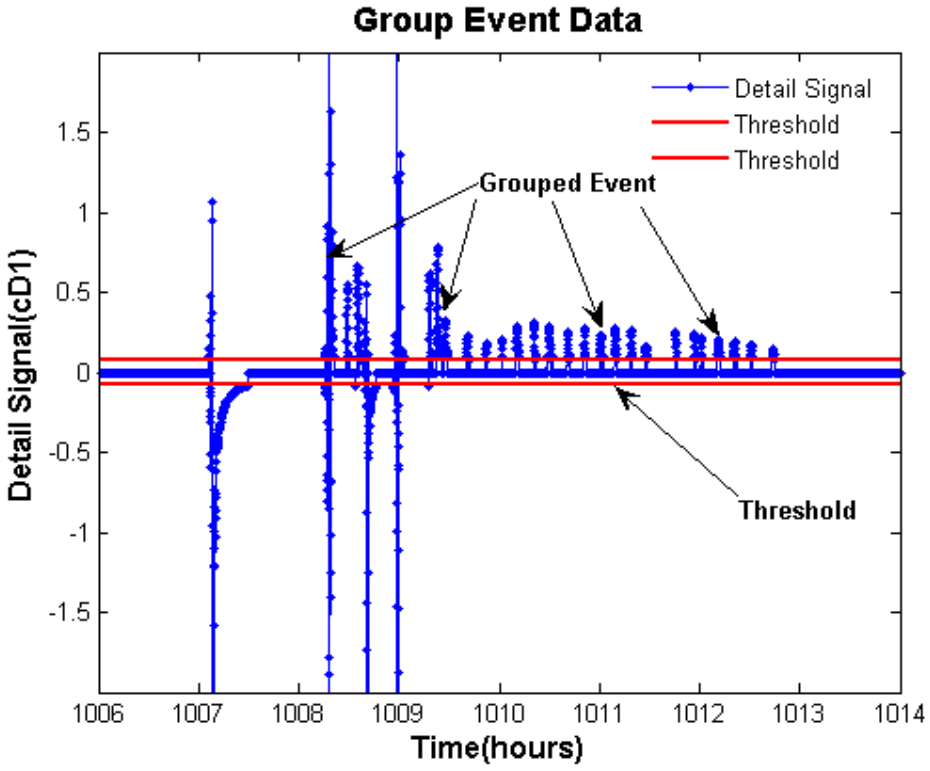


Figure 2.29 Zoom-In of Group event data of Example 1

After automatable selecting the threshold, the first condition is the minimal flow interval time, which is assigned as 5 minutes because 300 seconds is small enough for

two flow events. Hence the 5 minutes flow period is the maximum resolution of this algorithm. In figure 2.29, the grouped event is grouped with threshold.

The result of flow event identification is very sensitivity to the threshold value. The subsequence part will perform a sensitivity study for the threshold value. The higher the threshold first assigned to identify the flow event, the fewer the flow events identified. Figure 2.30 shows flow events detected with a high threshold with $\mu+10*\sigma$ or$\mu-10*\sigma$. The different color means a different flow period. The green one is the final flow period. From this figure, we can observe 7 flow events.

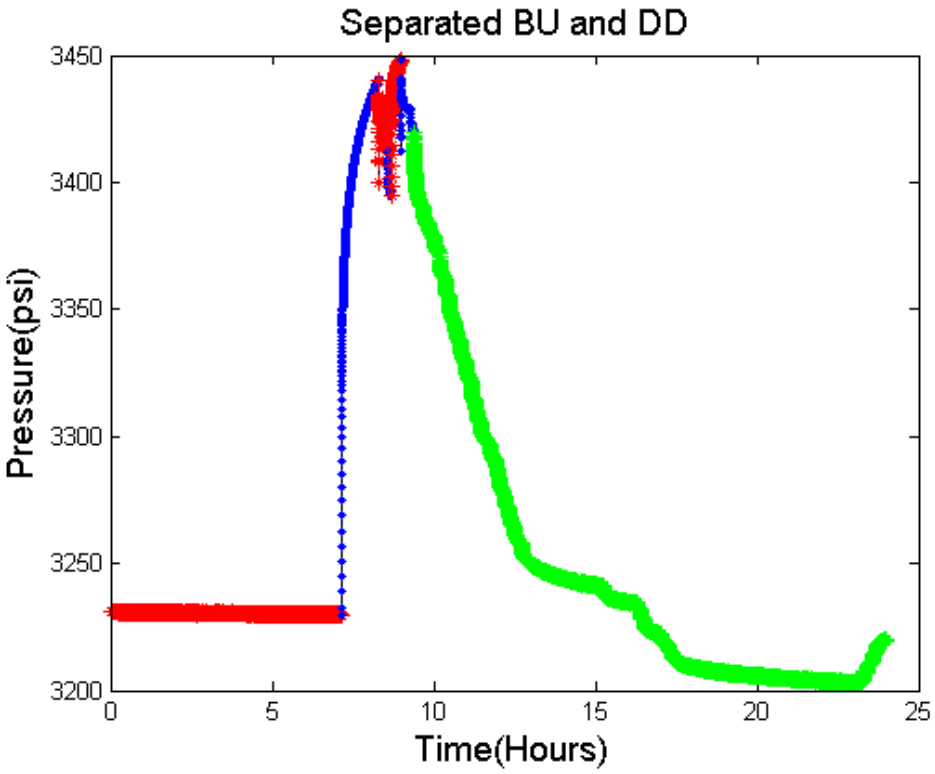


Figure 2.30 Flow event identification of high threshold of Example1

After automatic selection of the threshold, a smaller threshold is selected to identify the flow event. The new threshold is $\mu+1.53*\sigma$ or $\mu-1.53*\sigma$. Figure 2.31 shows the result with the optimum threshold. There are 39 identified flow periods which includes smaller DD flow periods.

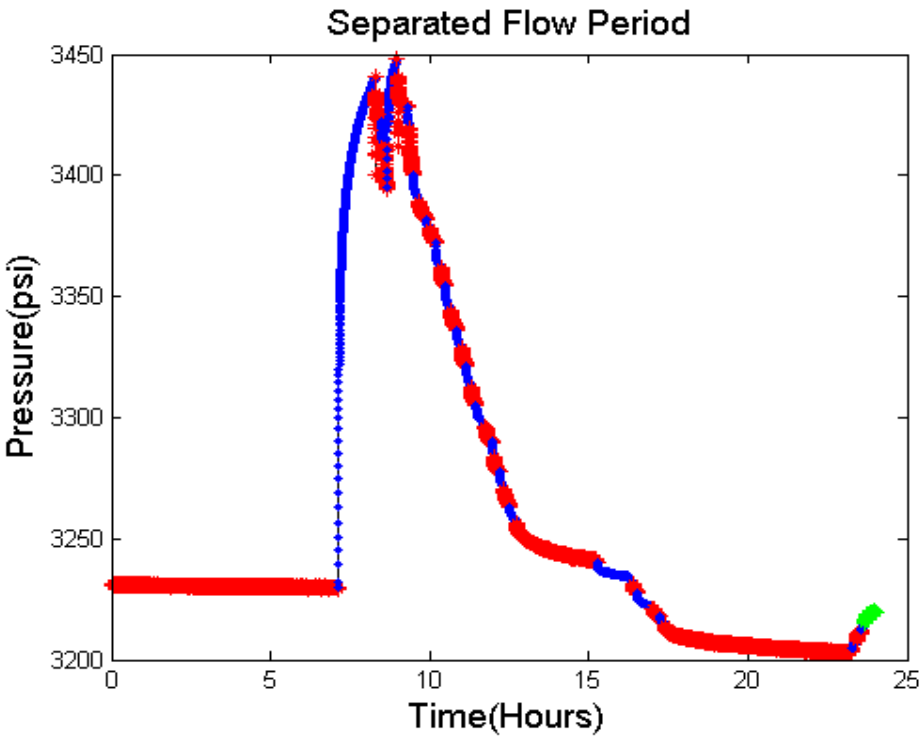


Figure 2.31 Flow event identification of optimum threshold of Example1

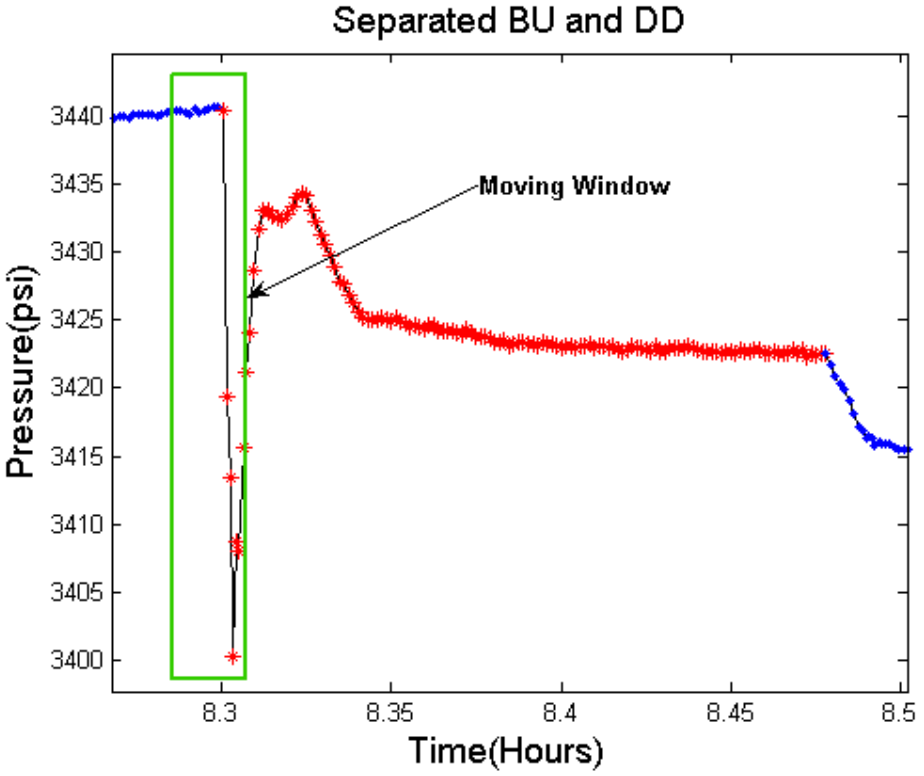


Figure 2.32 Refined break–point with moving window of Example 1

The previous step gives only an approximate region of flow event. After the breaking refinement, we can directly detect the right break point of flow period. Figure 2.32

shows the refined break point of the flow period. From the graph, we can confirm that the algorithm can find the correct break point for each flow period.

II. Field Example 2 –- PDG data with 1 second interval:

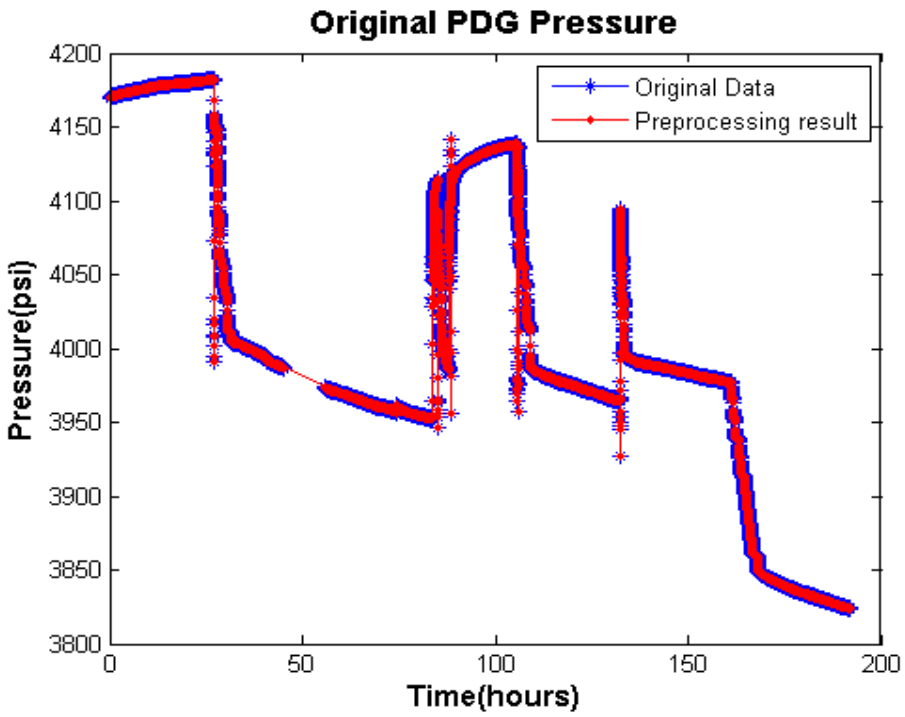


Figure 2.33 Original dataset and the result of preprocessing of example 2

This dataset comes from S well of the N-field. The well penetrated stacked reservoir sands saturated with oil, ranging in overall column thickness from 400ft to over 1,000ft. The oil from these zones is 35°-45° API gravity, with a very low sulphur content. The frequency of recording data is 1 second which is considered as the highest recording frequency of PDG pressure data. This dataset covers 8 days data and contains over 600,000 data points. There are three challenges in this dataset. The first challenge is that the dataset is very noisy because it records data every second. The second is that there is a large gap in the dataset. The third is how fast the algorithm can process these more than half million dataset. Figure 2.33 shows the original data set and the result of preprocessing. After the preprocessing step, the data interpolation, data decomposition, data evaluation and outlier removal steps are carried out for the dataset. Then the transient identification processes the results after outlier removal.

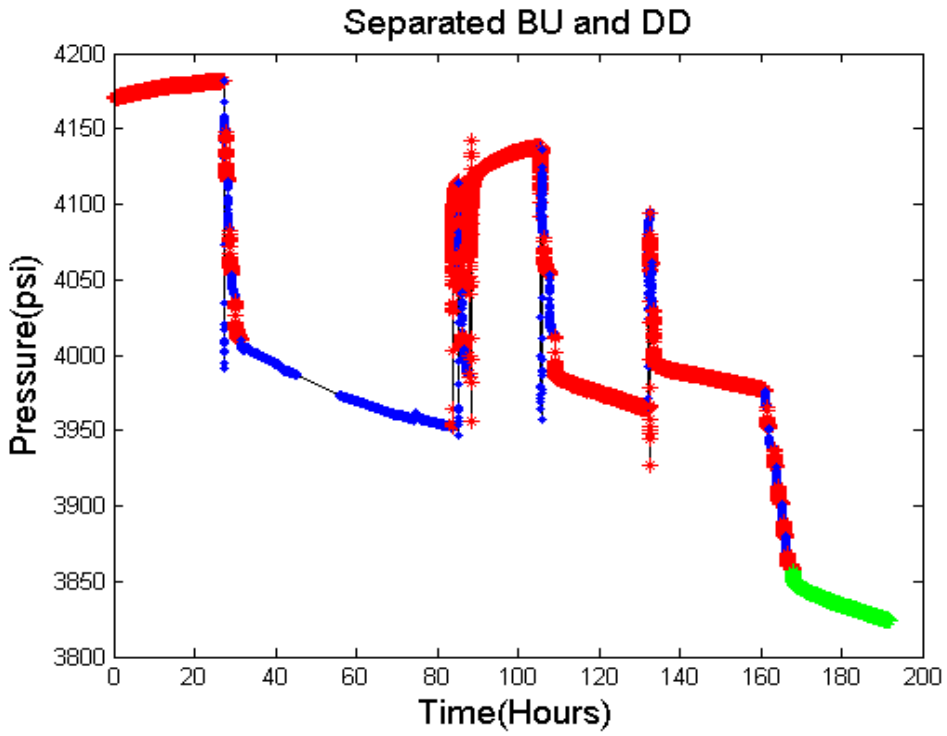


Figure 2.34 Flow event identification of Example 2

Figure 2.34, transient identification of Example 1, presents the result of the identified flow periods. There are three parameters which control the result of transient identification. We use only the default data recommended by the toolbox. The detail threshold is 0.1 which comes from the statistical calculation of detail information. The minus resolution of flow period is 0.01hours and the sample number is 5. The toolkits detect 47 flow periods in total. It takes only 10 seconds to identify the transient period using a laptop (CPU 1.7HZ, RAM 512M).

Figure 2.35 shows two phenomena. Firstly, there is unknown information at the beginning of second BU which is not flow event information. The algorithm treats this as noise information. The second phenomenon is the fluctuation at the beginning of the three DDs. The normal approach will treat this fluctuation as noise or as an outlier, but our algorithm will look at this fluctuation as information about the well and reservoir. Therefore, our approach retains all this information. Figure 2.36 zooms in on the later part of Example 2. This graph shows an 11 step DD which is caused by the increased flow rate. The duration of first 10 DDs is almost 0.5 hours before reaching the stable production rate.

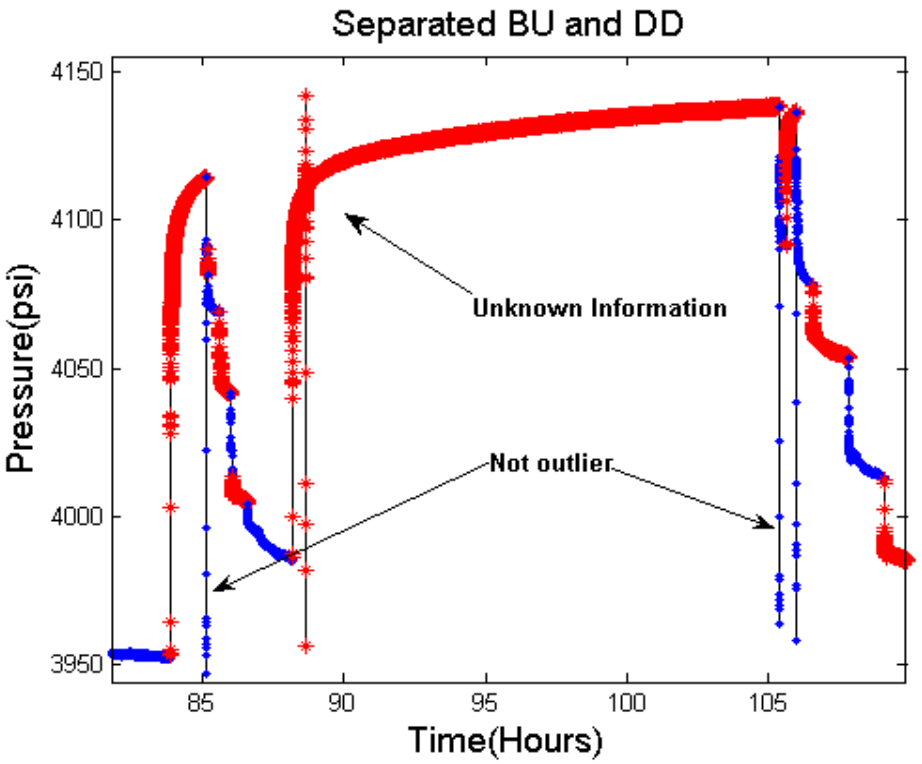


Figure 2.35 Zoom-In of Medial part of transient identification in Example 2

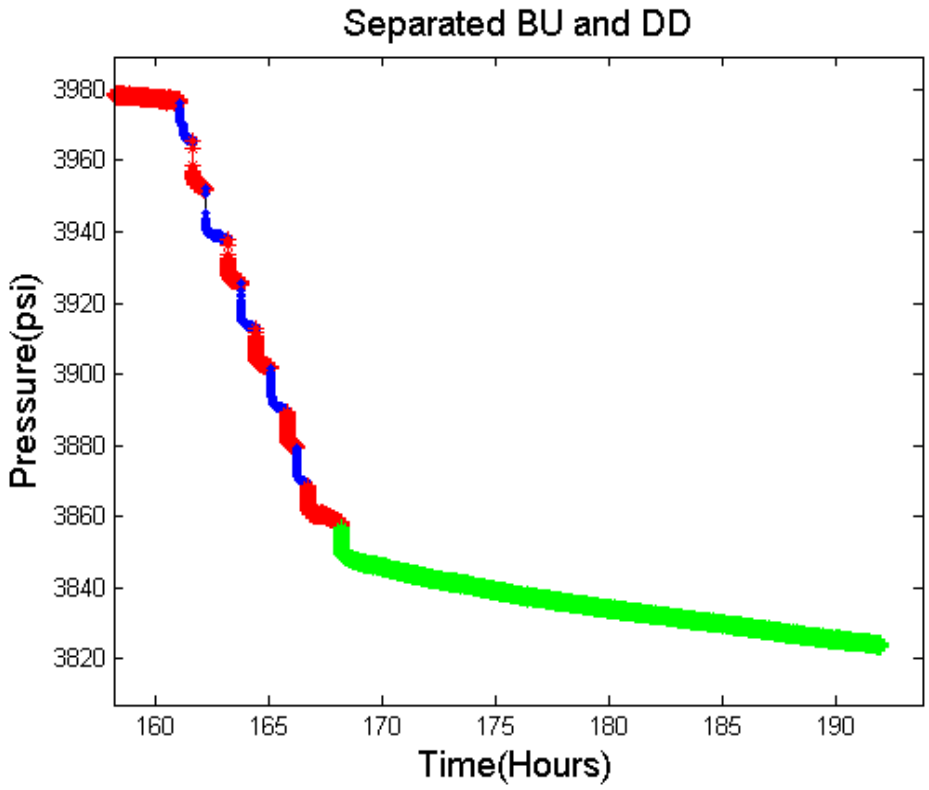


Figure 2.36 Zoom-In of Later part of transient identification in Example 2

III. Field Example 3 – PDG data with 5 seconds interval:

The dataset of example 3 is from the North Sea S-field which is introduced in data preprocessing part. This data contains three days PDG pressure data and records 51813 points. The interval between two points is 5 seconds. This example is used to test the ability of the algorithm in the high frequency PDG pressure. The challenge of this dataset is the dataset is that it is very noisy and the flow period is very complex. Figure 2.37 show the original dataset and the result of preprocessing. There are two marked parts which have a very complex flow period and do not resemble the normal BU and DD.

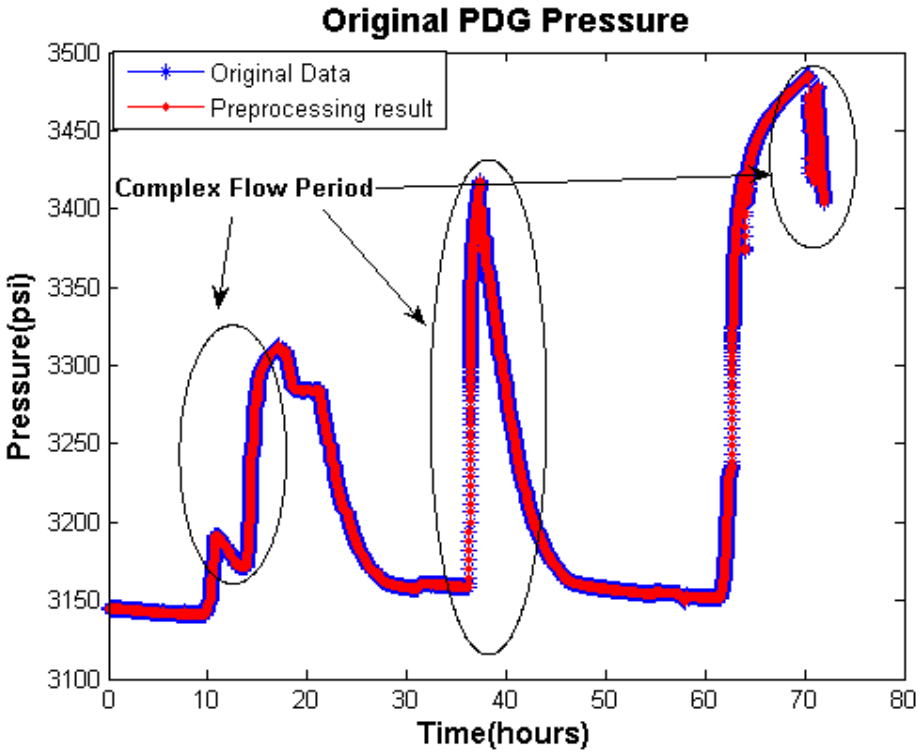


Figure 2.37 Original dataset and the result of preprocessing of example 3

Figure 2.38 presents the result of identified flow period. There are three parameters to control the result of transient identification. We only use the default data advised by the toolbox. The detail threshold is 0.031 which comes from the statistical calculation of detail information. The minus resolution of flow period is 0.01 hours and the sample number is 5. The toolkits detect total 90 flow periods.

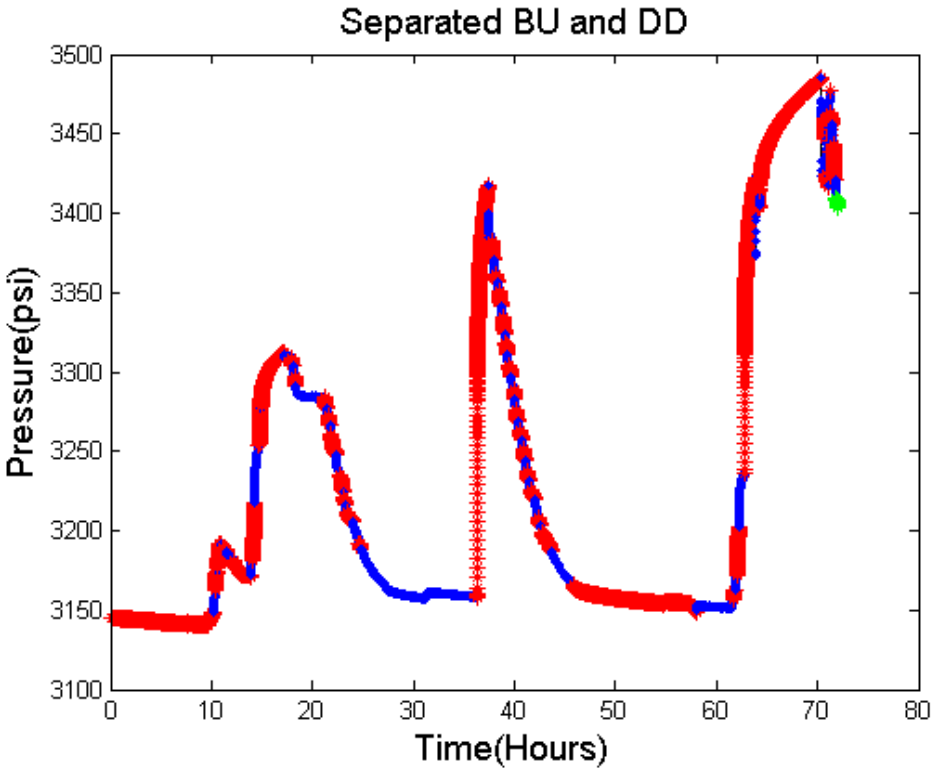


Figure 2.38 Flow event identification of Example 3

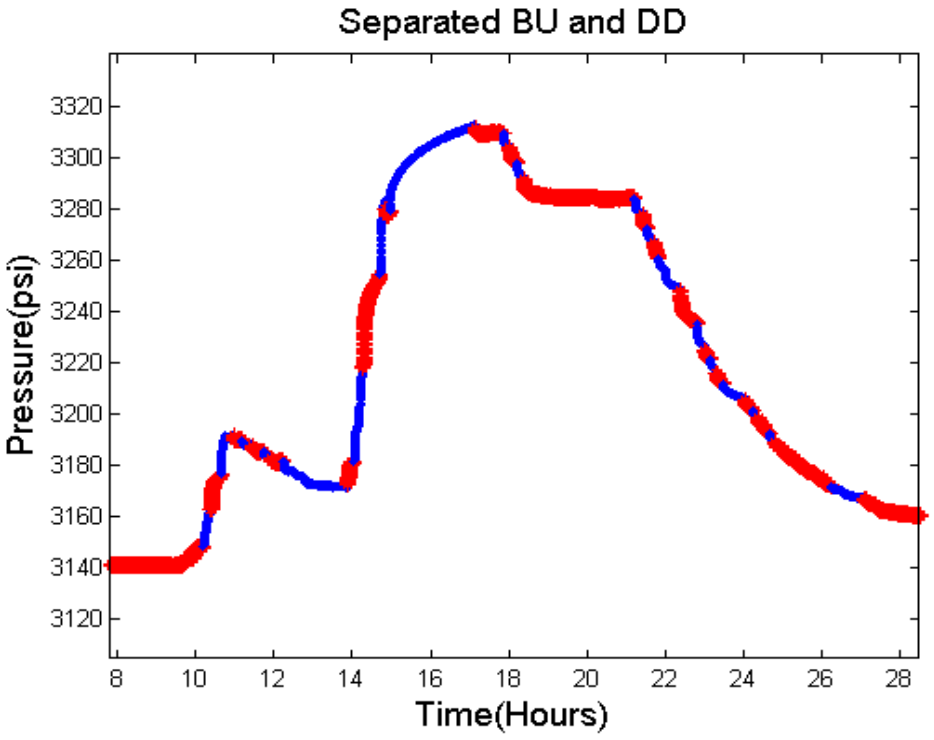


Figure 2.39 Zoom-in of the first part in Example 3

Figure 2.39 shows the complex flow period from 10 hours to 28 hours. There is a step BU during this period. We also can observe the same step BU between 60 hours and

70 hours. A multi-DD is followed.

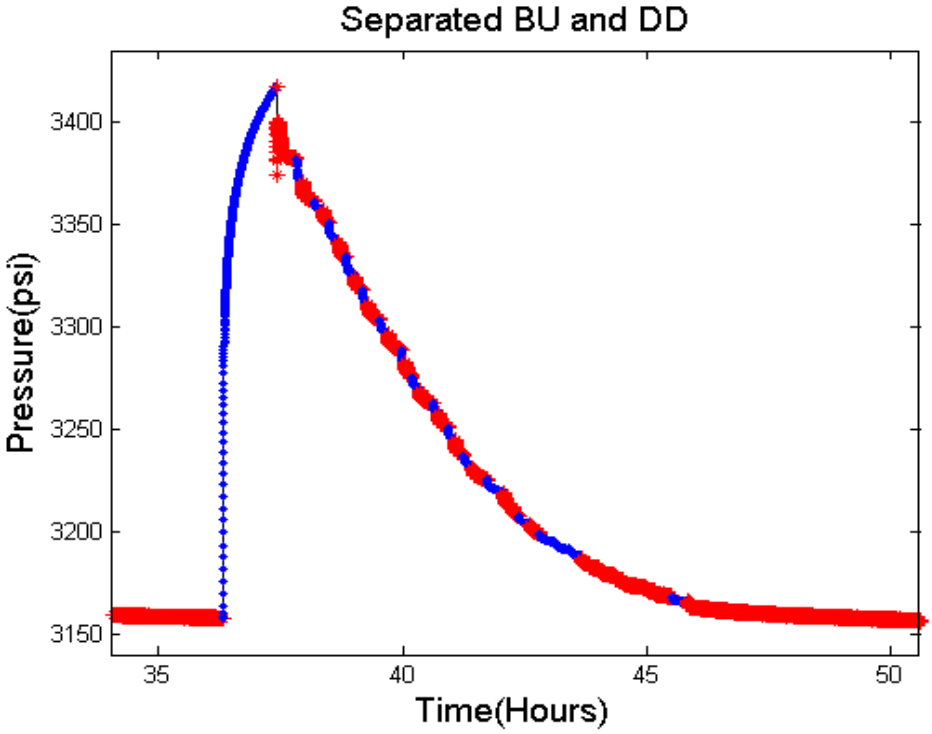


Figure 2.40 Zoom-in of the second part in Example 3

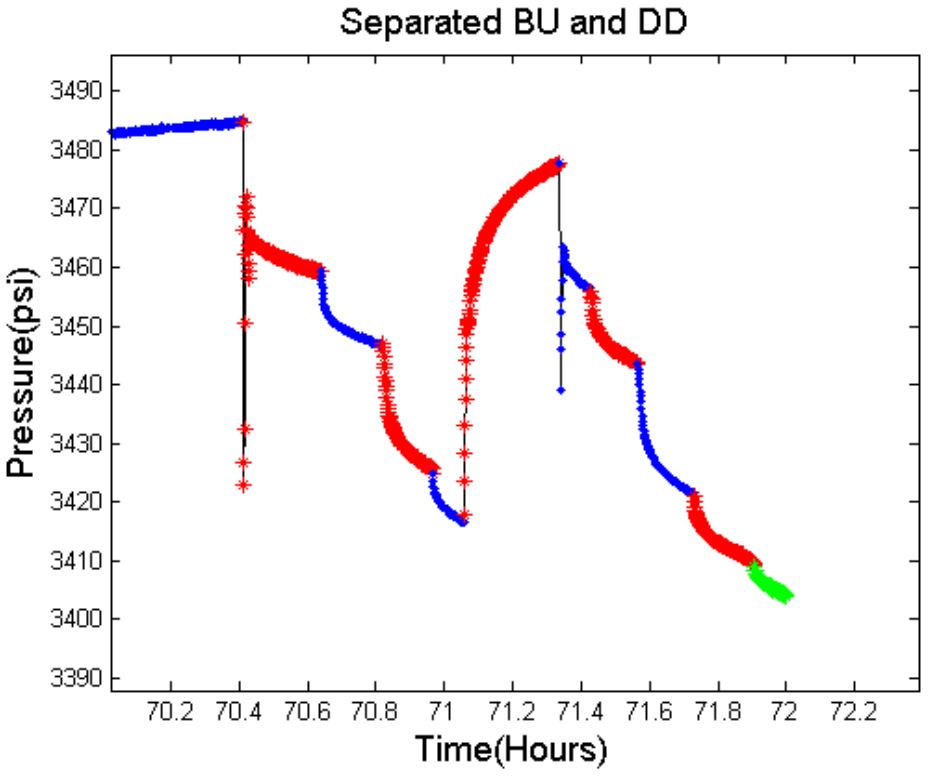


Figure 2.41 Zoom-in the third part in Example 3

Figure 2.40 illustrates a shut-in BU following 31 DD. Figure 2.41 shows the last part

of dataset which includes 1 shut-in BU following 4 DD and another BU following 5 DD. The break point of each period is the correct break point.

IV. Field Example 4 – mini-DST Example:

The dataset of example 4 is a mini-DST data with 1390 data points and a time interval of two points every 0.6 seconds. This dataset is from the software company to test the algorithm. The background of data can not be known. The whole dataset covers 14 minutes. The target of processing this mini-DST data is to identify the two main BU, marked in Figure 2.42.

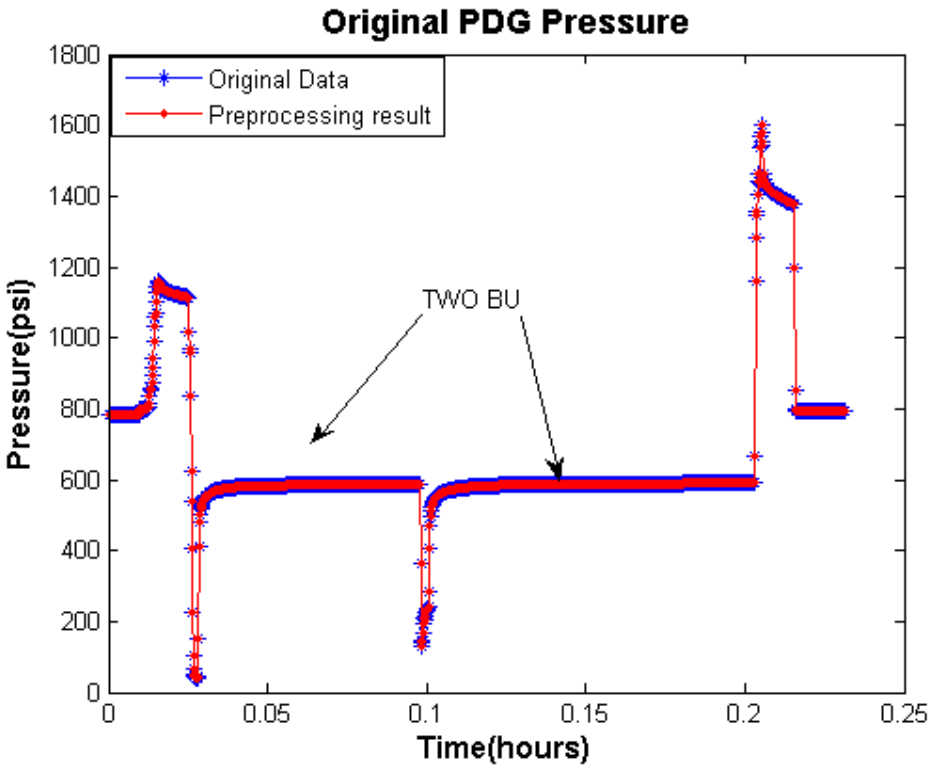


Figure 2.42 Original dataset of Example 4

The result of transient identification is shown in Figure 2.43, with flow periods divided. We use the default data advised by the toolbox. The detail threshold in this case is 2.3139 which comes from the statistical calculation of detail information. The minus resolution of flow period is 0.0002 hours and the sample number is 2. The main two BU are marked in Figure 2.43. There are 8 periods of the whole dataset. The two target BUs can be identified from dataset.

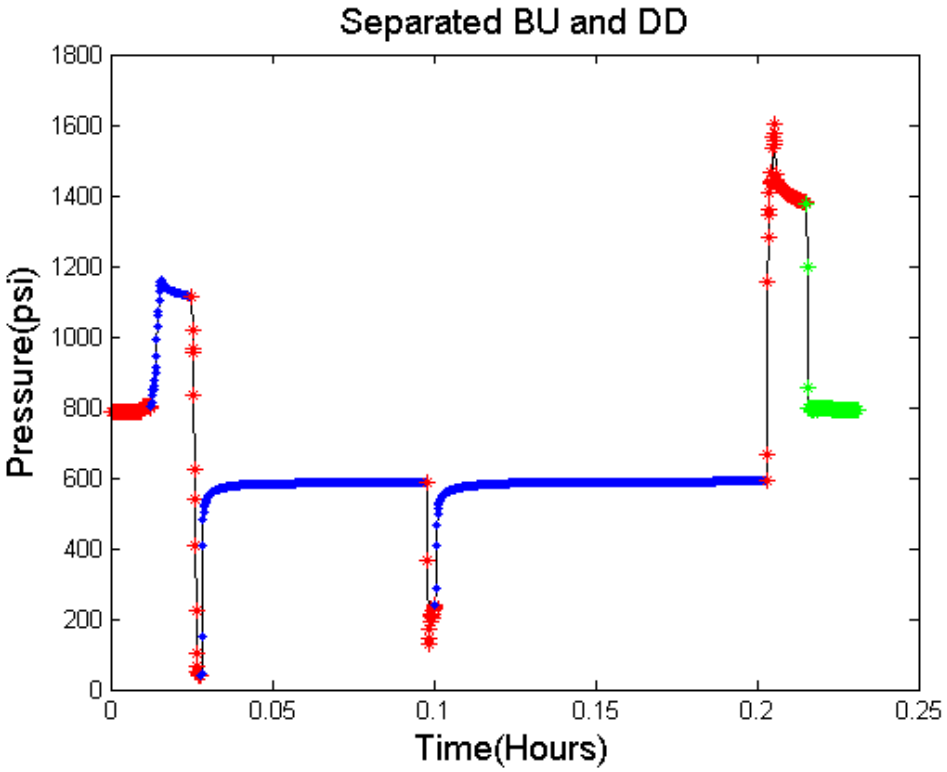


Figure 2.43 Flow event identification of Example 4

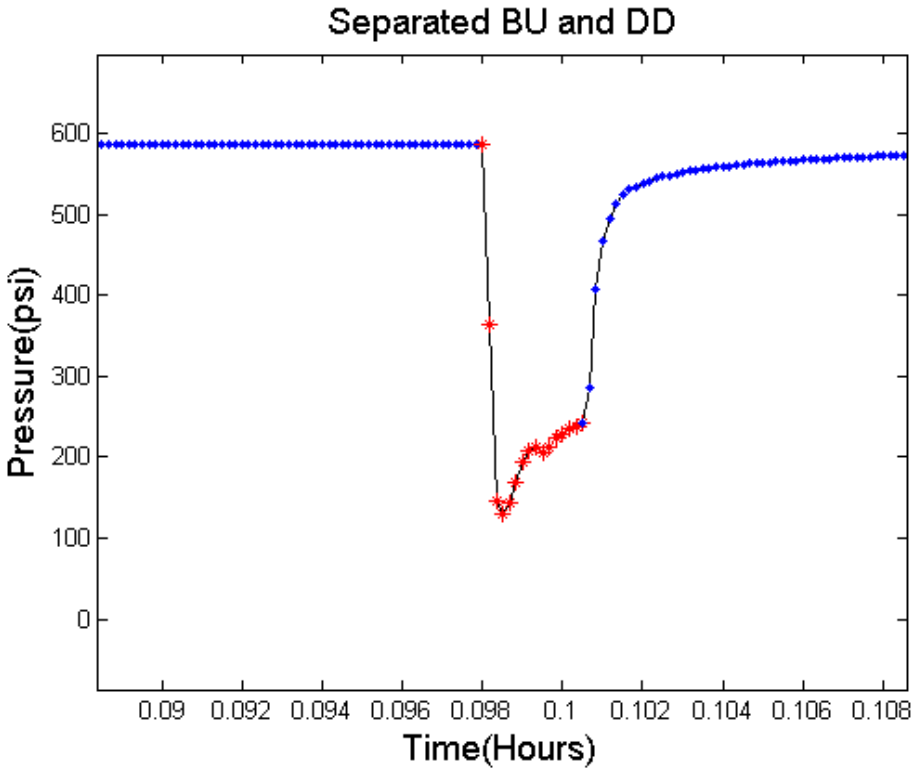


Figure 2.44 Zoom-in transient identification of Example 4

Figure 2.44 zooms in on the detail of the second main BU. The correct break point of

both BU and DD can be detected with this algorithm. Therefore, this algorithm also can be used to process mini-DST data.

V. Field Example 5 – large amount of dataset:

The dataset of example 5 is from the North Sea B-field as well. The main B reservoir, a thick channel sand, is estimated to contain recoverable reserves of 50 to 75 million barrels of light oil. The reservoir is developed with pressure depletion. This three month PDG pressure data records 1360106 points. The interval between two points is 5 seconds. Figure 2.45 shows the total dataset. This example is used to test the performance of the algorithm in processing a large scale dataset with more than one million points. The great challenge of this dataset is the large scale which can cause lower speed in displaying and calculation. The toolbox may even be destroyed by these huge data. The other challenge is the ability of toolbox to identify the entire flow event from this huge and complex dataset.

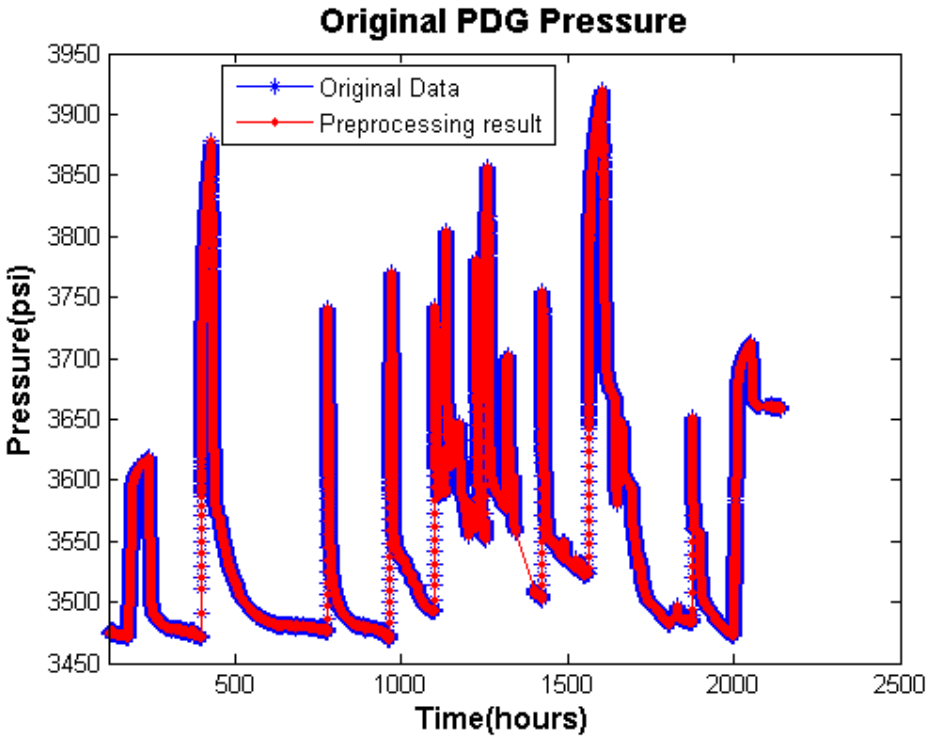


Figure 2.45 Original pressure dataset of Example 5

This toolbox is run on a laptop with 1.7 GHz CPU and 512 M RAM. Table 2.1 shows the performance of each step in the toolbox. As the table illustrates, the longest CPU time for calculation is about 1minute to identify the flow event and the longest CPU time for display is 15 seconds to display the flow periods. The total CUP time is 109.6597seconds. From this table, we can conclude that the performance of the

toolbox can meet the requirements of data processing.

Name of Step	CPU time for Calculation(s)	CPU time for Display(s)
Data processing	1.5508	7.5007
Data interpolation	5.3610	1.4327
Data decomposition	5.4993	0.3578
Data evaluation	0.2993	0.1578
Outlier removal	12.6412	1.1249
Identification of event	58.9026	14.8316

Table 2.1 CPU time of toolbox for Example 5

Figure 2.46 presents the result for the identified flow periods. There are three parameters controlling the result of transient identification. We use the default data advised by the toolbox. The detail threshold is 0.031 which comes from the statistical calculation of detail information. The minus resolution of flow period is 0.02hours and the sample number is 5. The toolkit detected 447 flow events.

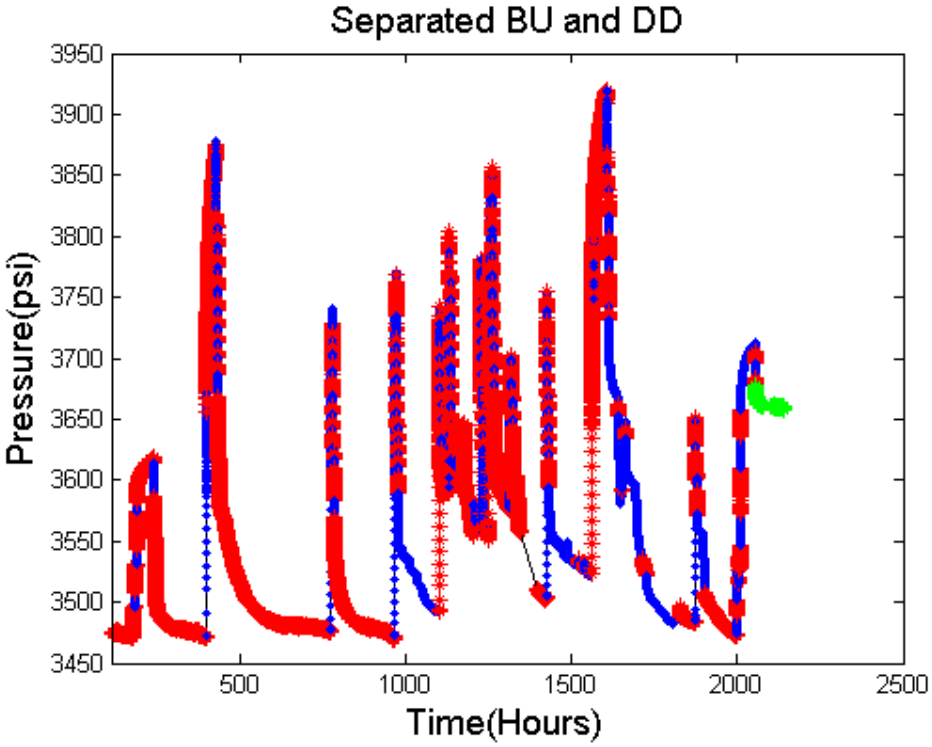


Figure 2.46 Flow event identification of Example 5

Figure 2.47 shows the complex flow period from 1095 hours to 1145 hours. There are two shut-in BUs, which can be used for well testing analysis during this period. The first point of every BU is detected very well. We also can observe small step DD after the two BU.

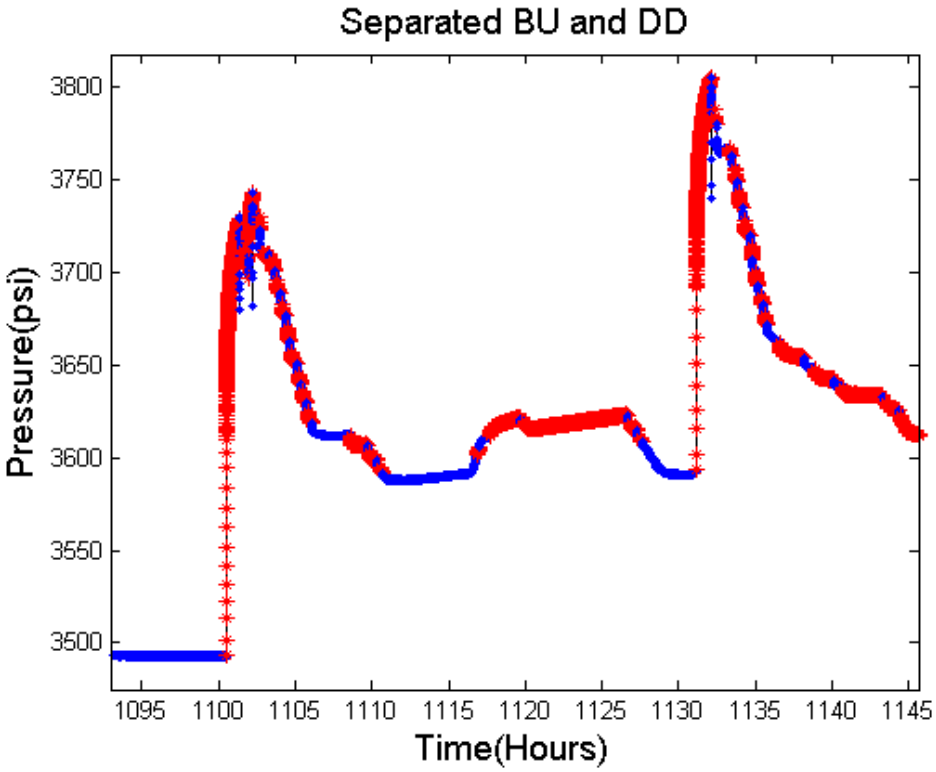


Figure 2.47 Zoom-in of flow event identification in example 5

2.7 Identification of the BU and DD

2.7.1 Identification of the BU and DD under three conditions

This step is to identify the BU and DD after flow event identification. Several approaches have been conducted to achieve this. Olsen finds the BU flow period with the help of the production history. He simply selects the BU period to perform the traditional well testing. Athichanagorn introduces flow history reconstruction to identify the flow period. Our method will consider three conditions to detect the BU and DD.

After flow event identification, the next step is to detect the BU and DD. The flow event is classified as three types of phenomena: DD, Shut-in BU and Rate-drop BU. Three conditions are considered for the detection of BU and DD. The first condition is when the PDG pressure is the only data information. This means the BU and DD should only be identified from the PDG pressure data. The pressure difference and the slope method can help to identify the BU and DD. However, this method has a problem in distinguishing the shut-in BU and rate-drop BU. The second condition is when user does not know the production history and only knows the downhole PDG

temperature and pressure. The temperature data are always recorded with the PDG pressure data. Temperatures have a relation with the production rate: when the rate remains constant, the temperature will be constant but when the production rate is zero, the temperature will decrease with time. Hence the flow information can be identified from the temperature. The third condition is when the production history, PDG pressure and PDG temperature are known information. Normally the first type of production history data is from downhole rate gauge which is very precise. The second type of production history data is the daily production rate data from the separator.

Every dataset has its limitations when identifying the BU and DD. The PDG pressure data can give the right break point, but can not distinguish the Shut-in BU and Rate-drop BU. The PDG temperature data can only give the shut-in BU information, but can not correctly give the other flow information. The production history suffers from the low resolution of dataset. In fact the best way is to perform cross checking to identify the BU and DD.

I. Identification of the BU and DD from PDG pressure

After identification of flow events, the beginning and end of every flow period are found. As we know, the pressure tendency of the DD flow period will go down from the beginning of DD flow period in the Cartesian plot. The pressure tendency of the BU flow period will go up from the beginning of the BU flow period. The algorithm is to compare the pressure on the both sides of grouped data. The grouped data is where the rate changed. The pressure on each side represents pressure when the rate is stable. Consequently, we can compare the pressure data on the both sides of the grouped data. If the preceding pressure is higher than the succeeding pressure, this flow period is a DD flow period. If the preceding pressure is smaller than the succeeding pressure, this flow period is a BU flow period.

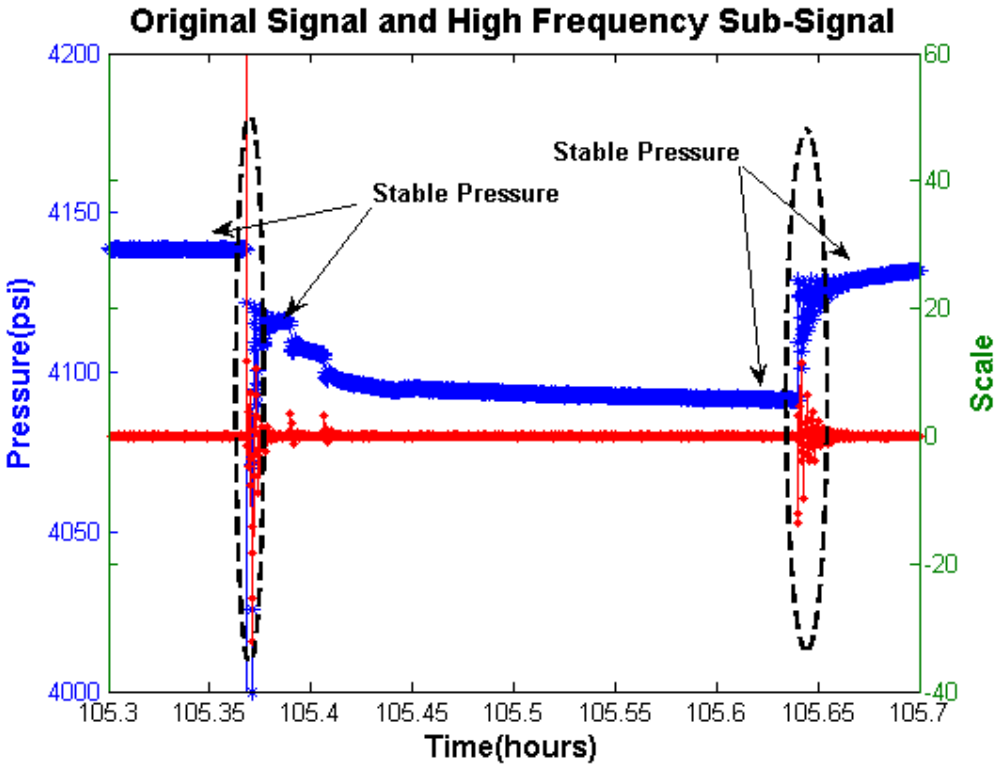


Figure 2.48 Identification of the BU and DD from PDG pressure in Example 2

Figure 2.48 show the logic of this method. The blue curve is the PDG pressure and the red curve is the high frequency signal of PDG pressure. After flow event identification, we group the pressure data when there is some violent change in the rate. From the first group, the flow period is DD and the second group is a BU. But in the algorithm, we did not compare the first point of the group data with the last point of the group data because there is large fluctuation at the beginning of the group data which may cause wrong identification of BU and DD. Hence we choose five data points before the beginning point of group and calculate the average for these five data. Then we choose five data points after the end of the group and calculate the average for these five data. Finally, we compare these two average points to come to a conclusion about the flow period. The above approach can easily identify the BU and DD, but cannot distinguish the shut-in BU from the rate-drop BU. However, the shut-in BU can be detected from the rate-drop BU in some situations.

II. Identification of the BU and DD from PDG pressure and temperature

It is very hard to distinguish the shut-in BU and rate-drop BU only from PDG pressure data. Therefore we consider obtaining more information from other data which can help to detect the shut-in BU and rate-drop BU. The PDG temperature sensor is normally installed with the PDG pressure sensor. This means that PDG temperature

data is recorded with PDG pressure at the same frequency and same time. This part explains how temperature data helps to detect the shut-in BU and rate-drop BU.

Both pressure and temperature are changing depending on downhole flow rate. The Joule-Thomson effect and frictional heating effects are the main dynamic factors causing flow bottomhole temperature to differ from the static formation temperature at that depth. The Joule-Thomson effect is when the temperature of fluids changes because of a pressure drop when a fluid expands at constant enthalpy. Therefore, during production, both the Joule-Thomson effect and frictional heating effects are the main factors causing sand face temperature to differ from the original reservoir temperature. For oil production wells, sandface temperature is always higher than original reservoir temperature, while for gas producing wells, sandface temperature can be either higher, lower or equal to original reservoir temperature due to both effects acting simultaneously. In the shut-in BU period, the early time temperature data are affected by wellbore storage in which there is a downhole flow rate from reservoir towards wellbore. During wellbore storage, flow rate decreases within a short period of time, approaching a zero value, causing the Joule-Thomson effect and frictional heating effects to vanish. When wellbore storage ends and there is no downhole flow rate, the bottom hole temperature starts cooling down over shut-in time because of the heat transfer between the well and the environment. In rate-drop BU period, the early time temperature will decrease because of the drop in the rate. The dropping rate reduces the effect of the Joule-Thomson and frictional heating effects. While the rate last longer enough to reach the steady state, the temperature will keep stable temperature. Figure 2.49 shows the difference between the two types of BU. One rate-drop BU and two shut-in Bus can be observed.

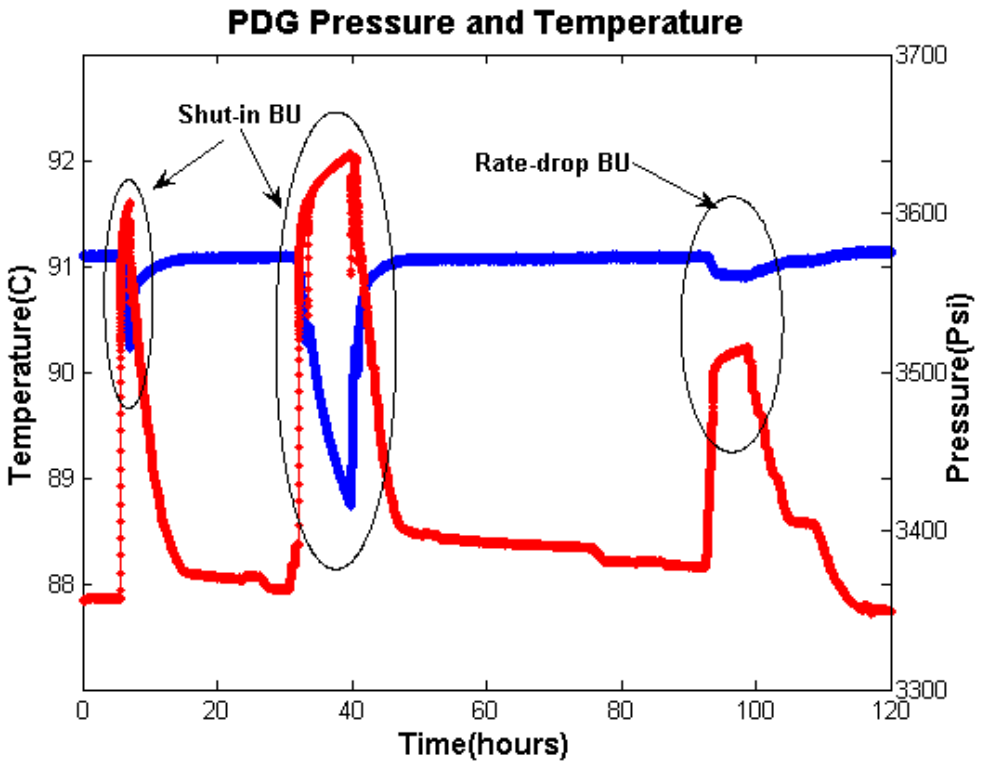


Figure 2.49 PDG pressure and temperature of Field Example 3

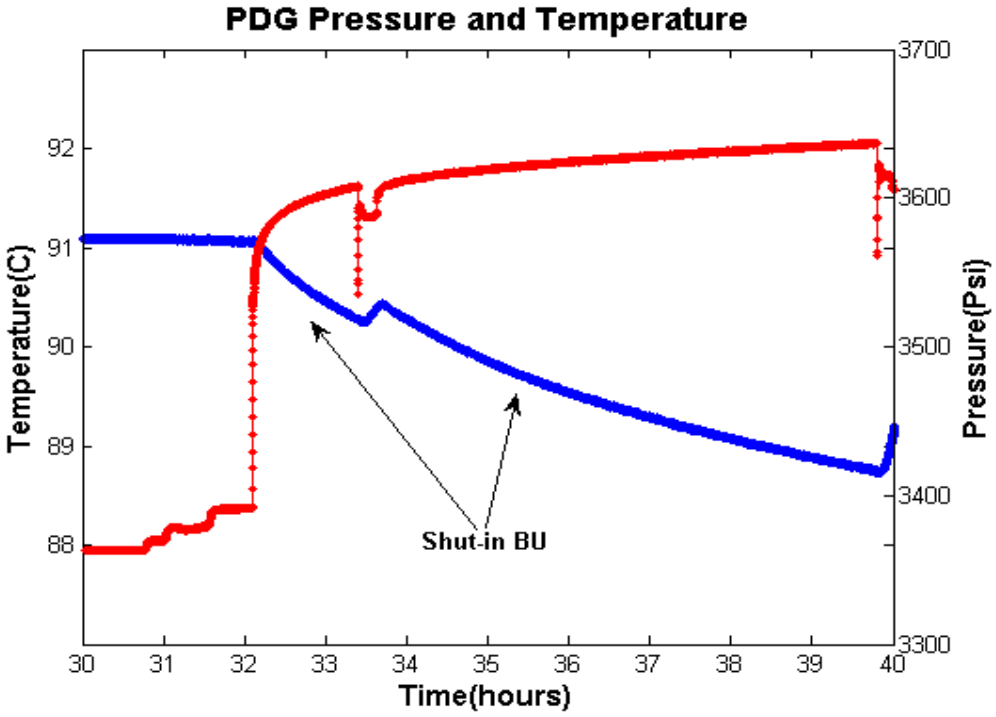


Figure 2.50 Part of PDG pressure and temperature (Shut-in BU) of Field Example 3

Figure 2.50 shows the decrease in temperature which is caused by the shut-in BU. We can observe that the temperature will keep dropping when the well is shut in. In

addition, figure 2.51 shows the temperature of the rate-drop BU, which is flat. The temperature remains stable when the rate-drop remains stable. So, when the pressure first drops and then remains stable, the rate-drop BU can be confirmed.

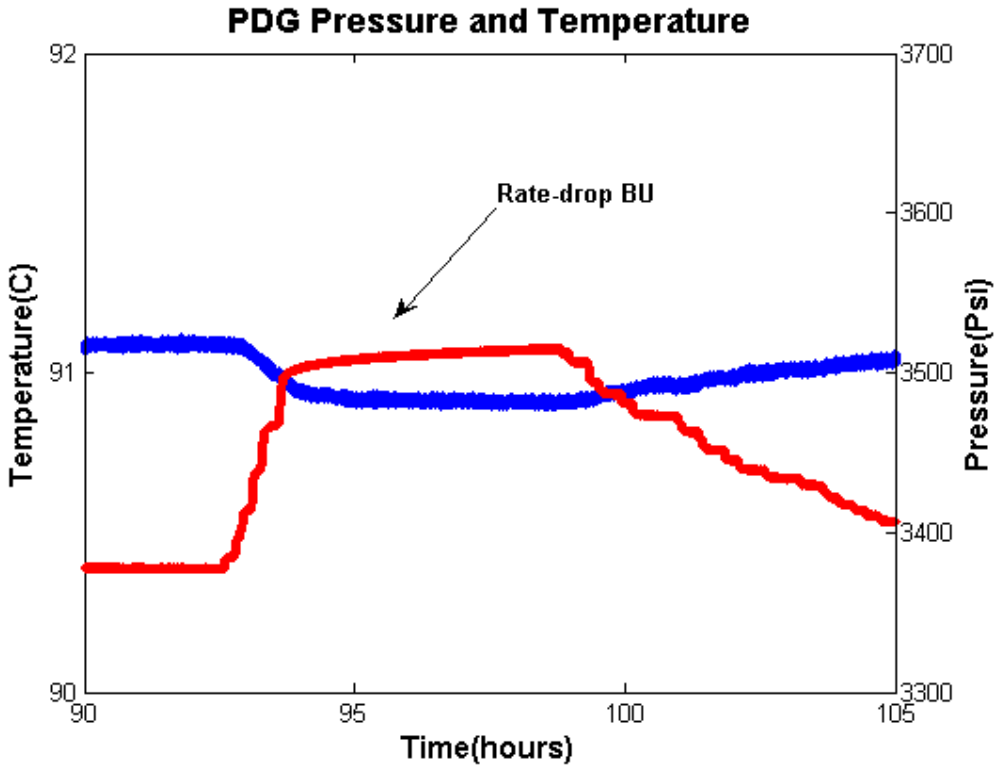


Figure 2.51 Part of PDG pressure and temperature (Rate-drop BU) of Field Example 3

III. Identification of the BU and DD from rate history, PDG pressure and temperature

The previous two sections introduced how to identify the BU and DD using the PDG pressure and temperature, which is easy to obtain. This section will introduce the use of flow rate information. Although most oil companies only install the downhole pressure gauge and temperature gauge, downhole flow meters are now installed by a few companies. The procedure to classify the flow period with the downhole rate data is very simple. The algorithm just needs to find the zero rate periods to decide the shut-in BU. However, the real rate is very complex because of the missing, inaccurate, noisy or sparse data. Figure 2.52 shows missing rate data between 5 hours and 20 hours. A zero value is also found in the DD period at 81 hours. There are also several small DDs in the large BU from 32 hours to 40 hours, but the rate only shows the zero value.

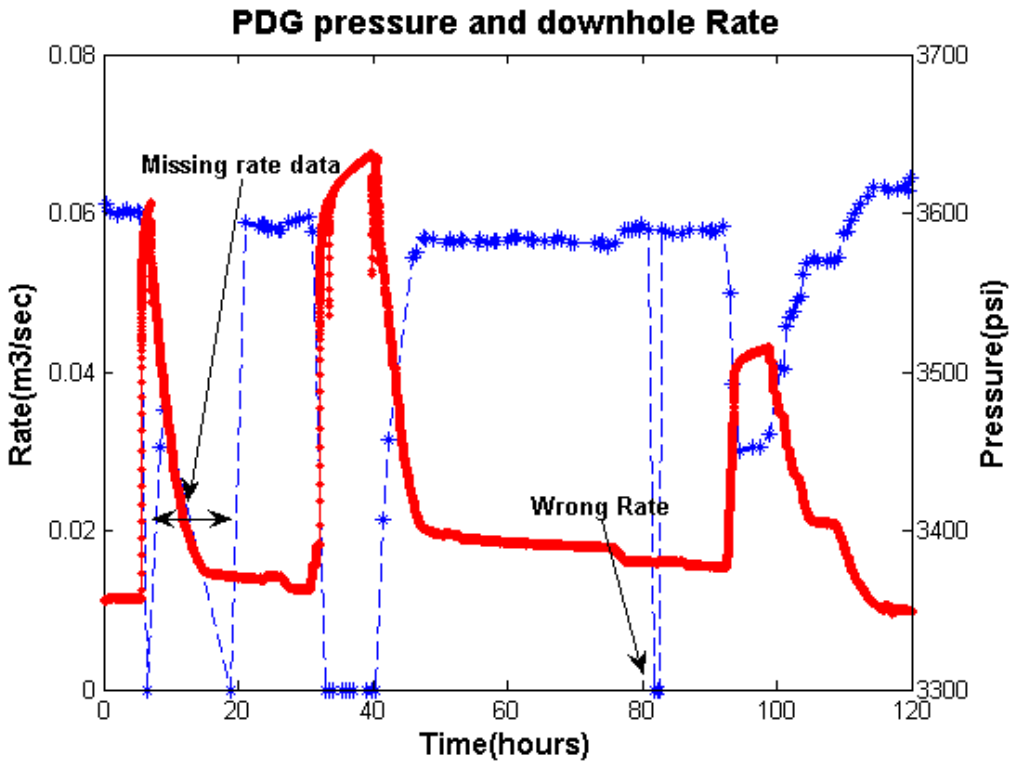


Figure 2.52 PDG pressure and downhole rate of Field Example 3

The best procedure to identify the BU and DD is to cross check the values of pressure, temperature and rate. The pressure data give the first estimate of BU and DD. Then temperature data is used to detect the shut-in BU. And the downhole rate history is compared with the results from pressure and temperature to confirm the final answer.

2.7.2 Identification of different phenomena during BU and DD

The PDG pressure data is very complex because this data integrates information from gauge, well, reservoir and workover. It is therefore essential to be clear about the PDG pressure data. Currently, the PDG pressure data is divided into different BU and DD flow periods. In order to obtain more information from the PDG pressure data, we go further to investigate the PDG pressure behavior. This step is to refine each flow period to obtain more information or to connect several flow periods to obtain the reasonable flow period. The following three behaviors are very common phenomena in the PDG data.

The first detected phenomena are multi-step DDs. Normally; we think there is just one DD when the well is opened. But the real situation is that the well is opened gradually. The second phenomenon is a degree of fluctuation in the BU and DD. This fluctuation

may be caused by the operation, or by fluid phase variation. The third phenomenon is that the BU and DD pressure do not follow the general trend. This mainly happens in the low frequency part. This may be caused by well interference or a changed flow boundary. These phenomena are detected to help the user to understand and analyse the pressure data. The current algorithm can consider only ,these phenomena but real situations are more complex than these. It is necessary to integrate other information, for example, geological, fluid, workover and production information as well as drilling information from other engineers.

I. Identification of the multi-DD

The ideal DD is a flow period in which the rate changed from one value to another value. This ideal DD data can be explained with current well test theory. But in the real world, the production rate did not directly increase to the production value because this operation would damage the well and cause safety problems. Therefore the operator opens the well gradually. This procedure may last one or two hours.

Figure 2.53 shows a shut-in BU followed by 30 small DDs. This multi-period DD can help us to check or recover the right rate history. If we want to analyze this DD data, it does not make any sense to analyze every single small DD. We need to consider all the multi-DD as a whole DD flow period. Then the superposition method or deconvolution approach can be applied to analyze these data. Figure 2.54 shows the whole DD.

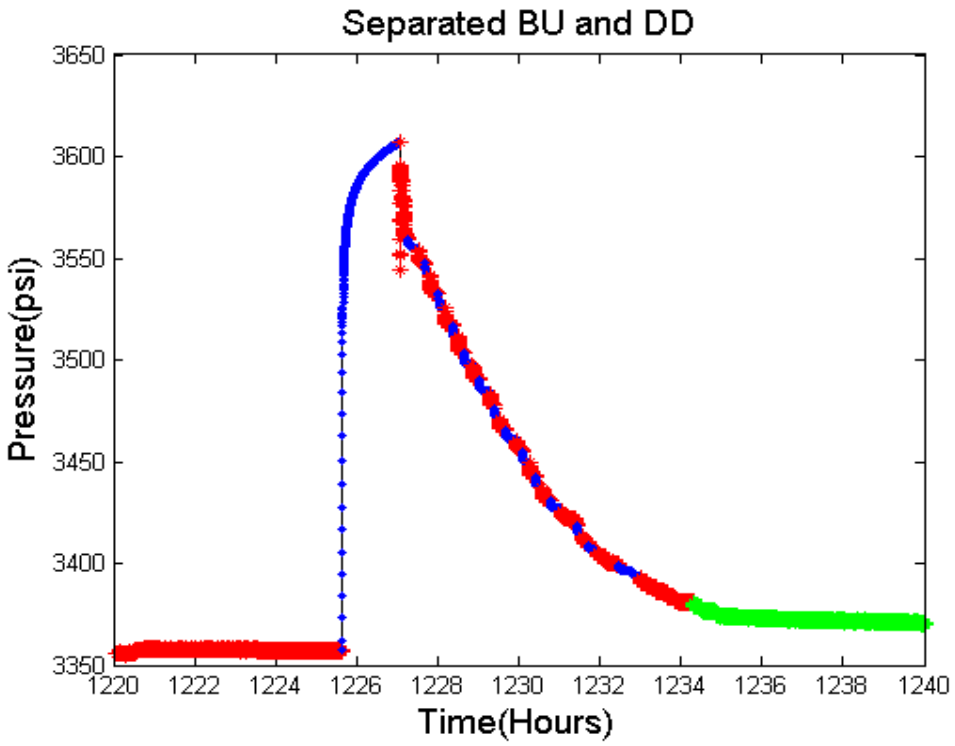


Figure 2.53 Separated multi-DD of Field Example 1

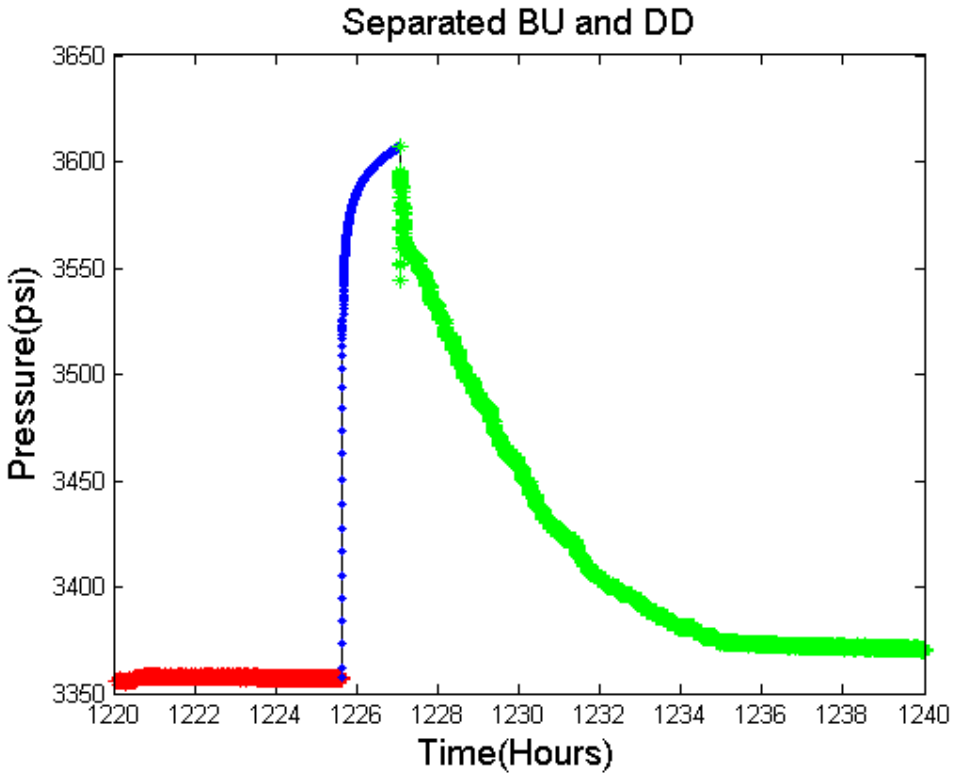


Figure 2.54 Connected multi-DD of Field Example 1

Identification of the fluctuation during BU and DD

The second phenomenon is some fluctuation which occurs during BU and DD. This

fluctuation may be caused by some operation and always occurs in the high frequency part of signal. This fluctuation is identified as the beginning of a BU or DD in flow event identification part. The fluctuation breaks the main BU into several small Bus, of which the first BU can be analyzed and the rest cannot be used. Figure 2.55 shows a fluctuation in the main large BU which was broken into two small BU of Field Example 2.

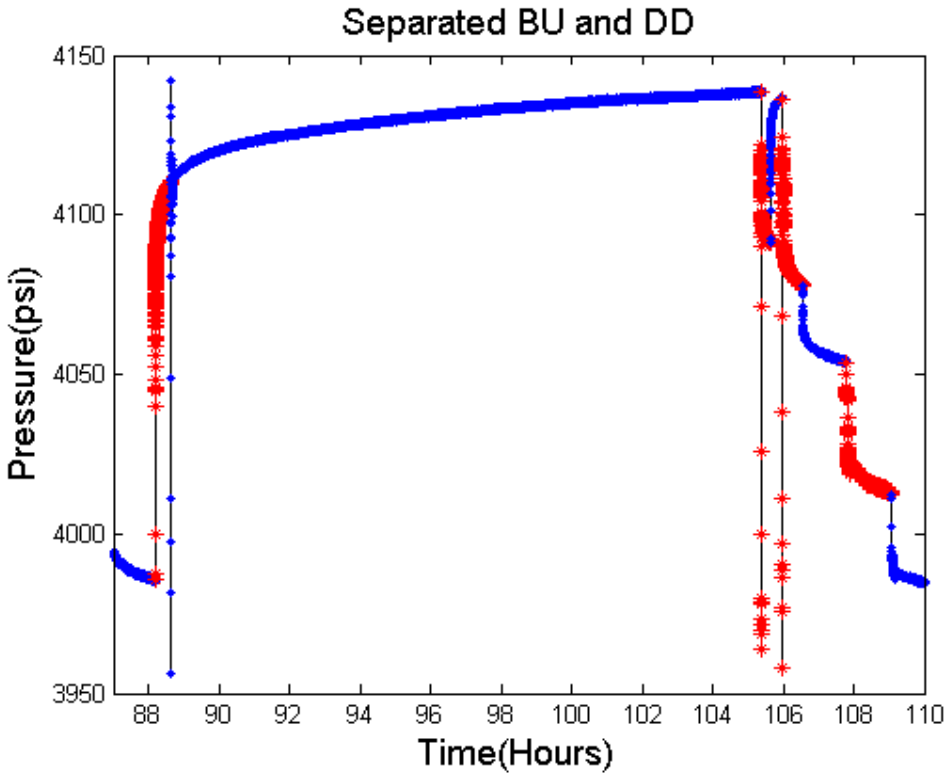


Figure 2.55 Fluctuations during BU Field Example 2

From other information like temperature and rate, we can confirm these two BUs are, in fact, one main BU. Accordingly it has been connected as one main BU in figure 2.56. Since the fluctuation is noise, it will be filtered in the further processing stages.

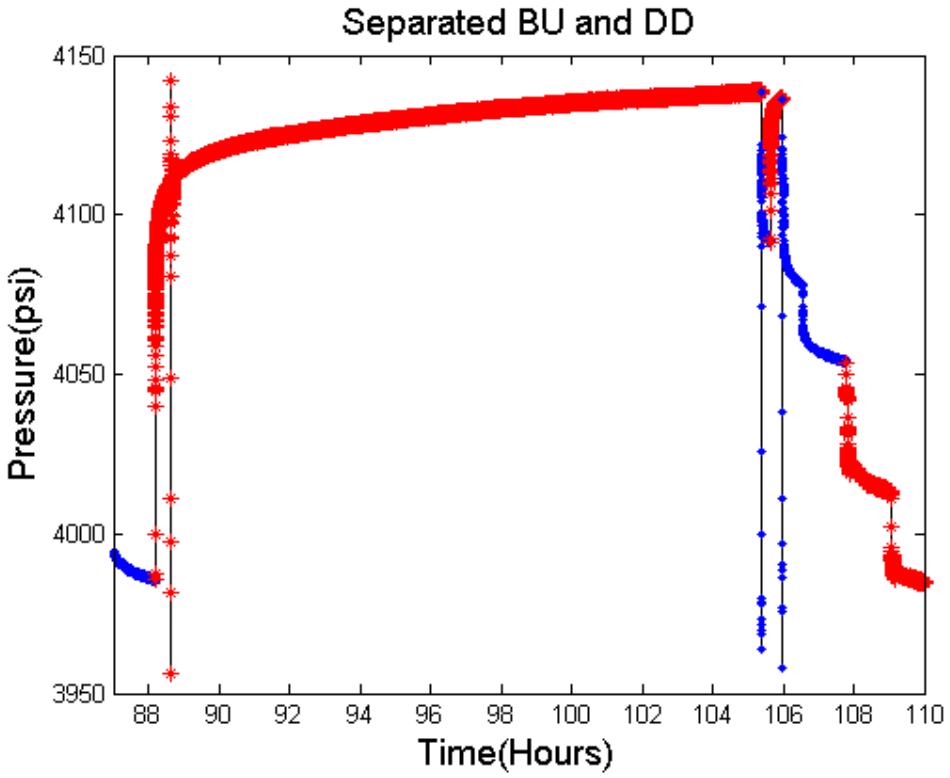


Figure 2.56 Fluctuations in one BU Field Example 2

II. Identification of low frequency information during BU and DD

After identifying the high frequency information during BU and DD, this sub-section focuses on the low frequency signal in BU and DD. The low frequency signal is the information from the reservoir or information from other wells. The trend of normal BU will go up and the trend of normal DD will go down but some abnormal trends are also found. Figure 2.57 shows a very complex flow period of real PDG pressure data. This dataset is from the other well of W Field. The reservoir lies at a depth of between 1,768m and 2,728m. It consists of good-quality tubiditic sands of a Tertiary age. The W reservoir is a low-relief NW/SE trending anticline, which is 9km long and 4km wide. It consists of a 15m oil leg, overlain by a gas cap and it contains 68 million barrels of oil. The graph shows a multi rate-drop BU from 100 hours to 120 hours followed a shut-in BU. The shut-in BU can be identified with the PDG temperature information in Figure 2.58. There is some abnormal behavior in DD and shut-in BU which can be identified in the low frequency part of signal.

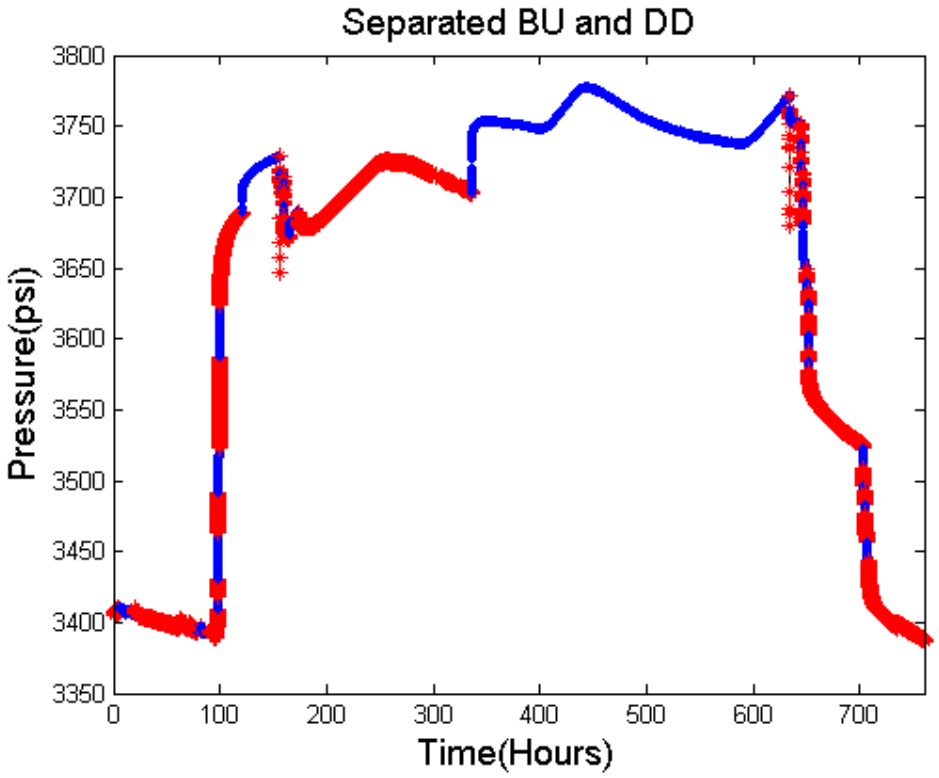


Figure 2.57 Identified low frequency flow event of W field dataset

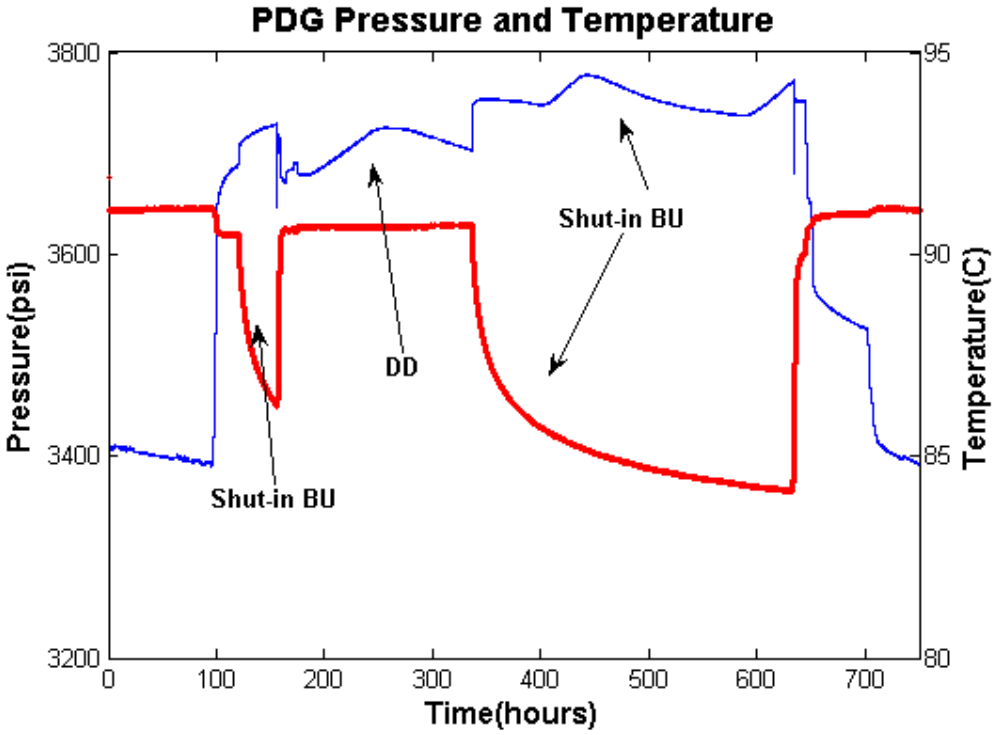


Figure 2.58 Identifying BU and DD from pressure and temperature of W field dataset

Figure 2.59 shows the second shut-in BU and the maxima line. These maxima lines

indicate that there is some smooth singularity. This abnormal behavior may be caused by interference from other wells or variation of properties in the reservoir.

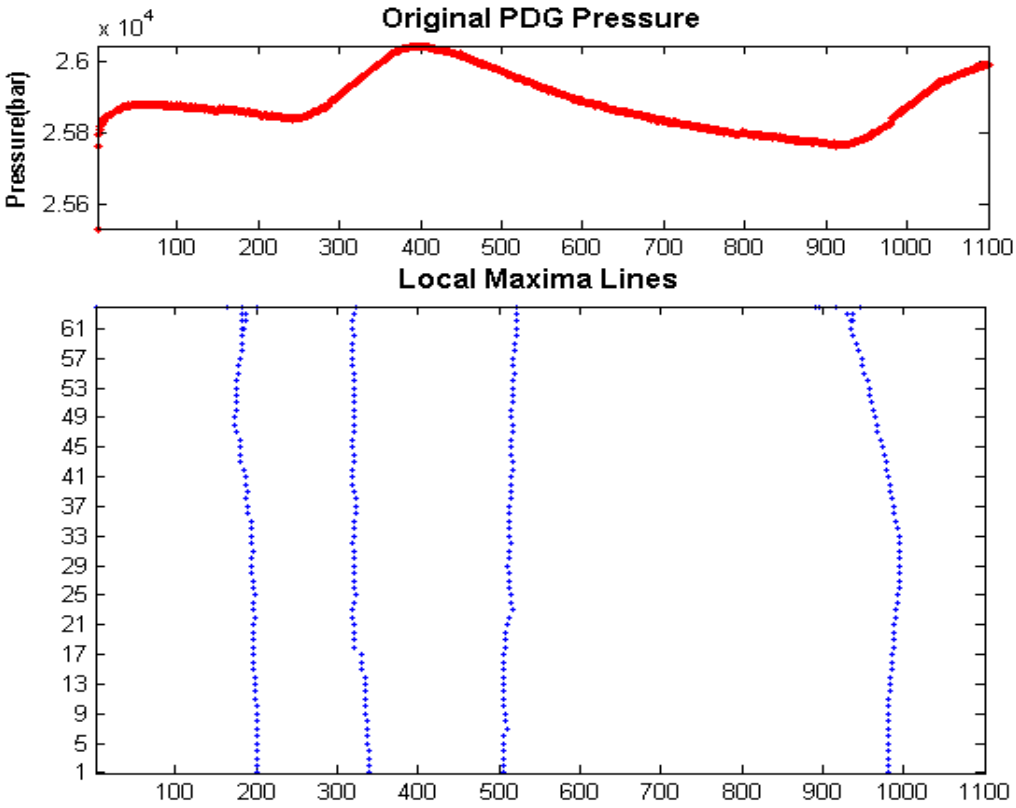


Figure 2.59 Identification of low frequency information during BU of W field dataset

2.8 Data denoising and data reduction

After processing the PDG data, the long term PDG pressure data is divided into different flow periods. We will focus on a single flow period. These data are from original data which is noisy and is recorded in high frequency. It is essential to further process the data to obtain clean data from the following analysis. The further processing includes two modules: denoising and data reduction. This part will discuss how to realize the denoising and data reduction in a single flow period and in the whole dataset.

In signal processing or computing, noise can be considered as data without meaning, that is, data that is not being used to transmit a signal, but it is simply produced as an unwanted by product of other activities. The noise can be caused by the PDG gauge, the well, or other environmental factors. The wavelet method is applied to perform denoising. The wavelet method first decomposes the signal into different level sub-signal. The high frequency signal is considered as denoised. The high frequency

signal will be set to be zero and the sub-signal is reconstructed to recover the original signal. The denoise approach is used for denoising the whole data set. It is found there are some smearing effects at the beginning and end of every flow period. Therefore we advise denoising for every flow period. This also avoids over-smoothing the signal. Denoising has an important role in the well test setup procedure.

The PDG pressure may be recorded for each second so the size of the data is huge. The first target of data reduction is to reduce the number of data points while retaining the whole trend of the dataset so as to plot or view the dataset more easily and quickly. The new approach is to use the wavelet decomposition to find the parts with sharp changes information. The second step is to retain more of the dataset during the sharply changing part and select less data to represent the stable change part. The other target of data reduction is to reduce data for each flow period. Normally it is sufficient to have one hundred points, which can represent the information about the reservoir model in a log-log plot. The approach is to sample the original flow period data with an equal scale to the log-log plot.

2.8.1 Denoising

I. Algorithm of Denoising

The noisy signal model is basically of the following form: $s(n) = f(n) + \sigma e(n)$ where time n is equally spaced. Normally, the noise level is supposed to be constant in the whole dataset. The de-noising objective is to suppress the noise part of the signal s and to recover function f.

From the view of wavelet transform, noise is a small, high frequency fluctuation in the data and is easily located by the corresponding fluctuations in the detail signals at low levels of decomposition. Several articles have presented the wavelet method known as wavelet shrink to denoise. The de-noising procedure principles can be summarized as follows:

1. Decompose

$$Y = W(S) \tag{2-9}$$

Choose a wavelet; choose a level, N. Compute the wavelet decomposition of the signal S at level N.

2. Threshold detail coefficients

$$Z = \delta(Y, \lambda) \tag{2-10}$$

For each level from 1 to N, select a threshold and apply a soft threshold to the detail coefficients.

3. Reconstruct

$$S = W^{-1}(Z) \tag{2-11}$$

Compute wavelet reconstruction using the original approximation coefficients of level N and the modified detail coefficients of levels from 1 to N. Where S is the signal, W is the wavelet transform, δ is the threshold operator, λ is the threshold value and W^{-1} is the inverse wavelet transform.

The idea behind wavelet shrinkage is that only a few large wavelet coefficients originate from the signal while the rest and small ones originate from noise, meaning that one can denoise the signal by shrinking all the small coefficients towards zero and keeping the large ones.

II. Denoising Synthetic Example

Figure 2.60 present the procedure of denoising the signal. The red curve is the original synthetic data. This original signal is first discomposed into 3 Level signals with wavelet sym7. The sub-signal can be found in the left part of Figure 2.59. Then the fixed form threshold is selected as the threshold which plot in the figure with yellow dashed line. The threshold is applied for the detail signal for Level 1 to level 3. Then, the denoised detail coefficients are plotted in the right side of Figure 2.59. The last step is to reconstruct the signal with denoised detail coefficients and Approximation coefficients. The yellow curve in the right part of plot is the denoised signal. We can found the noise has been compressed in the front part of signal.

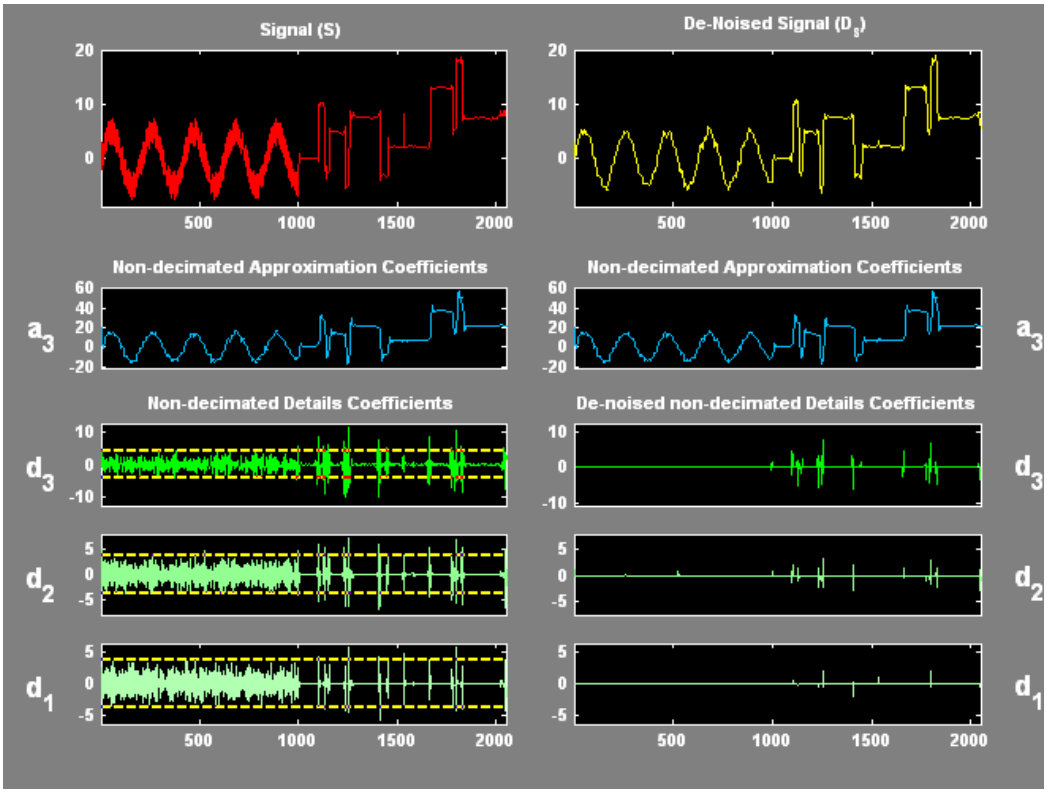


Figure 2.60 denoising the synthetic Example in Wavelet theory section

In the wave shrink method, several parameters need to be determined in order to remove the noise efficiently: noise estimation; level estimation; wavelet type; shrinkage rule; threshold value. The subsequence part will perform some sensitivity study for the synthetic Example.

The first sensitivity study will compare the denoised result with different wavelet. Figure 2.61 shows the different denoised data with different wavelet. This study tested five kinds of wavelet to denoise the synthetic examples dataset with fixed form threshold and 5 Levels decompositions.

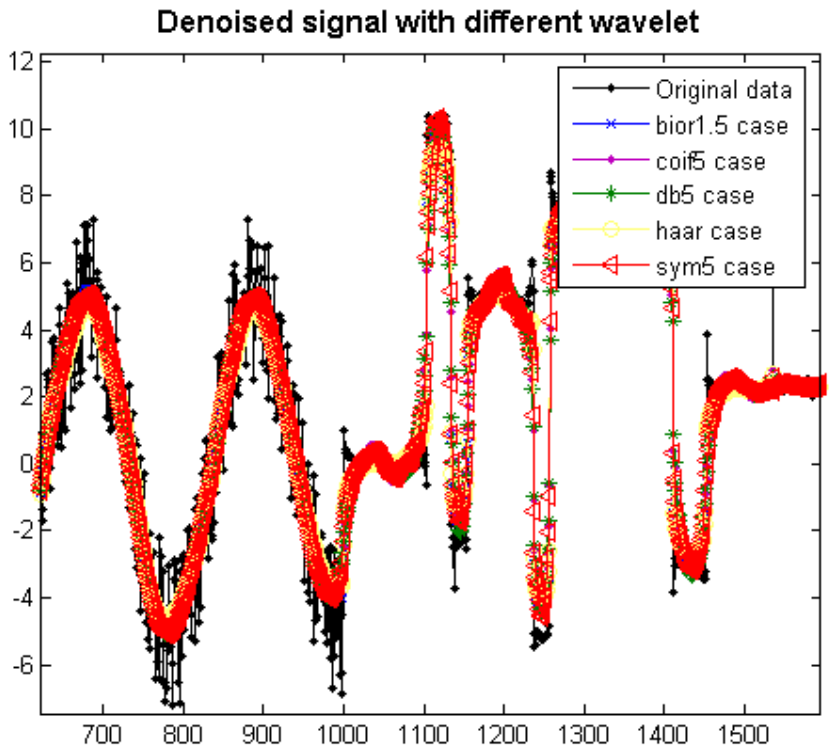


Figure 2.61 Comparison of denoised data in wavelet sensitivity study

	Standard Deviation	Median absolute Deviation	Mean absolute deviation
haar wavelet case	1.071	0.5229	0.7576
db5 wavelet case	1.226	0.5045	0.8641
sym5 wavelet case	1.193	0.5348	0.8506
coif5 wavelet case	1.22	0.5536	0.8775
bior1.5 wavelet case	1.039	0.4764	0.7345

Table 2.2 Comparison of denoise residuals in wavelet sensitivity study

The denoise residuals is used to evaluate the result. The denoise residuals is equal to the noised signal minus the denoised signal. Table 2.2 shows the comparison of evaluation parameter of denoise residuals. From the standard deviation of denoise residuals, the db5 wavelet case have the biggest value. The db5 wavelet case will be considered as the best denoised case. However, the db5 wavelet may be not the best wavelet for data denoise. So, different dataset should do wavelet sensitivity study for denoise.

The second sensitivity study will compare the denoised result with different wavelet level to decompose signal. Figure 2.62 shows the denoised signal with different

decomposition level. This study tested four kinds of decomposition level to denoise the synthetic examples dataset with the same db5 wavelet and same threshold approach. From figure 2.62, we can found the level 4 case may be the best denoised data.

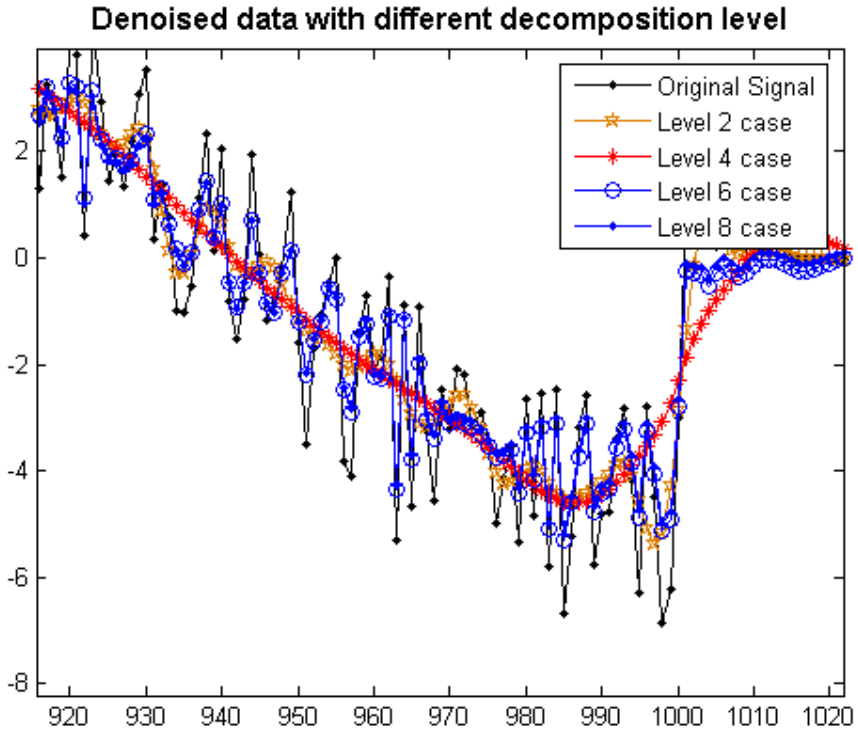


Figure 2.62 Comparison of denoised data in different decomposition level

Table 2.3 shows the comparison of evaluation parameter in the sensitivity study of decomposition level. The level 4 case have the best performance of three parameters in table 2.3. The conclusion can be made that the denoised data with higher decomposition level may not have the best denoised result.

	Standard Deviation	Median absolute Deviation	Mean absolute deviation
Level 2 case	0.9326	0.2324	0.5745
Level 4 case	1.16	0.4419	0.7801
Level 6 case	0.581	0.258	0.416
Level 8 case	0.5901	0.2672	0.4279

Table 2.3 Comparison of denoise residuals in decomposition level sensitivity study

The third sensitivity study will focuses on the threshold which is the most sensitivity parameter for wavelet denoise approach. There are four thresholds Selection rules will be compared.

'minimaxi' thresholds method uses a fixed threshold chosen to yield minimax performance for mean square error against an ideal procedure. The minimax principle is used in statistics to design estimators. Since the de-noised signal can be assimilated to the estimator of the unknown regression function, the minimax estimator is the option that realizes the minimum, over a given set of functions, of the maximum mean square error.

'heursure' is a mixture of the two previous options. As a result, if the signal-to-noise ratio is very small, the SURE estimate is very noisy. So if such a situation is detected, the fixed form threshold is used.

'sqtwolog' uses a fixed form threshold yielding minimax performance multiplied by a small factor proportional to log(length(s)).

'rigrsure' uses for the soft threshold estimator a threshold selection rule based on Stein's Unbiased Estimate of Risk (quadratic loss function). You get an estimate of the risk for a particular threshold value t. minimizing the risks in t gives a selection of the threshold value.

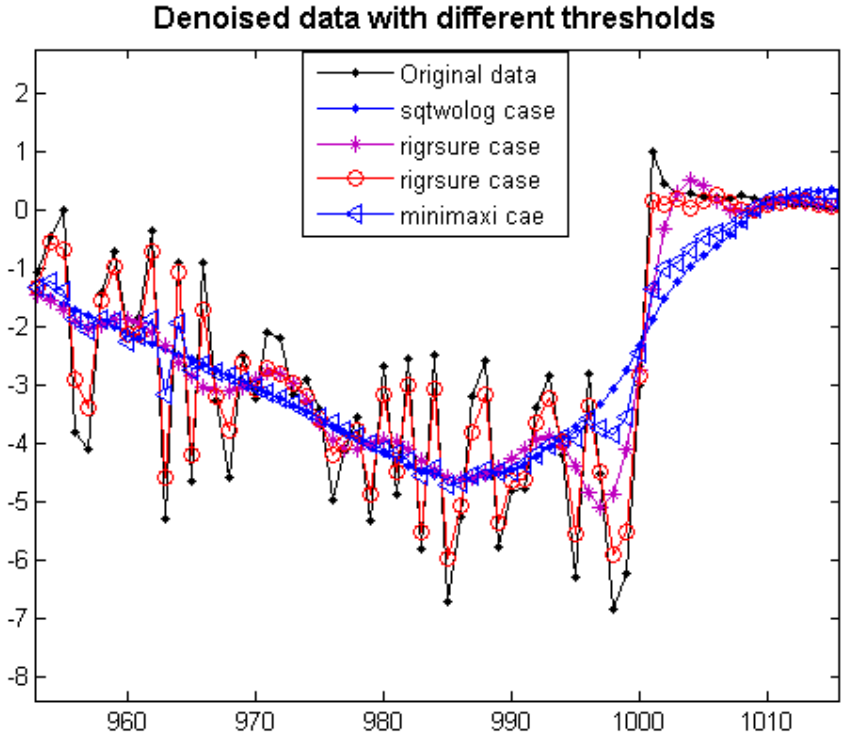


Figure 2.63 Comparison of denoised data in different threshold approaches

Figure 2.63 illustrates the denoised data with different threshold approaches. From the figure, we can find sqtwolog approach give the best denoised result in the stable part of

dataset. While, the rigrsure have the best denoised result when the dataset is fluctuation. From table 2.4, we also can conclude that the sqtwolog approach give the best value of three parameters.

	Standard Deviation	Median absolute Deviation	Mean absolute deviation
'sqtwolog' case	1.16	0.4419	0.7819
'rigrsure' case	0.3612	0.1256	0.2464
'heursure' case	0.9403	0.2305	0.5899
'minimaxi' case	1.002	0.3578	0.682

Table 2.4 Comparison of denoise residuals in different thresholds

From this case we can find the noise level is not constant in the whole dataset. As a result, it is essential to give a variable threshold for denoising. Since the threshold for noise can not be a constant data value, the whole dataset is considered to be divided into different parts. Each small part has a constant threshold approach.

III. Field Example of denoise

Figure 2.64 shows the denoised pressure of field Example 2. The whole dataset is denoised with db5 wavelet, sqtwolog threshold approach and Level 8 decomposition signal.

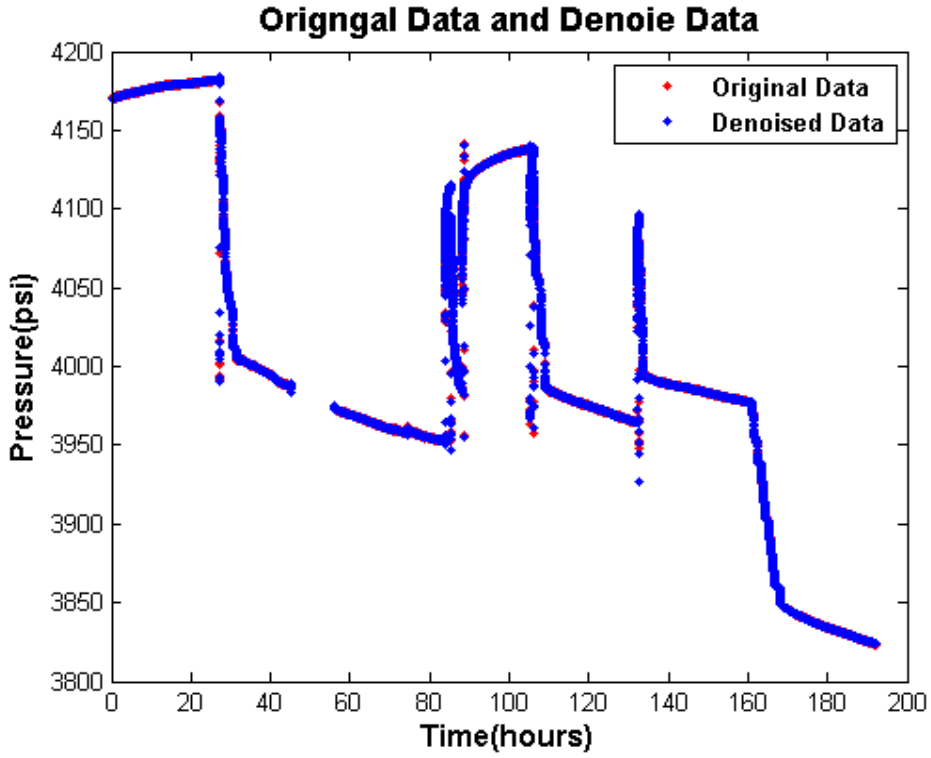


Figure 2.64 denoising the whole dataset of Field Example 2

After the data denoising of the whole dataset, we found denoised data have more fluctuation at the beginning and end of a flow period. This phenomenon called smear effect. The smear effect can be seen from figure 2.65. All current denoising approaches tend to smear out sharp features at the beginning or end of flow period because the sharp change occurred at these points. So, this study will consider denoising the separated flow period. The threshold method used in this work for denoising. The soft threshold method is used in the continuous data regions, and the hard threshold method is applied to the data located in the vicinity of the discontinuities. Figure 2.66 shows the smear effect have been removed.

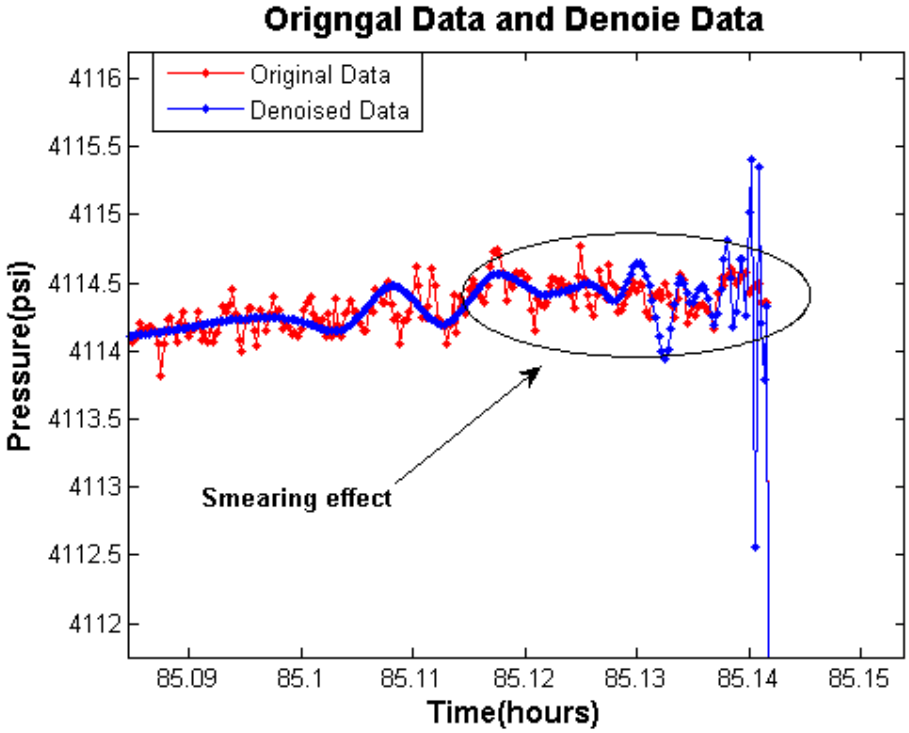


Figure 2.65 Zoom-In of the smearing effect in Field Example 2

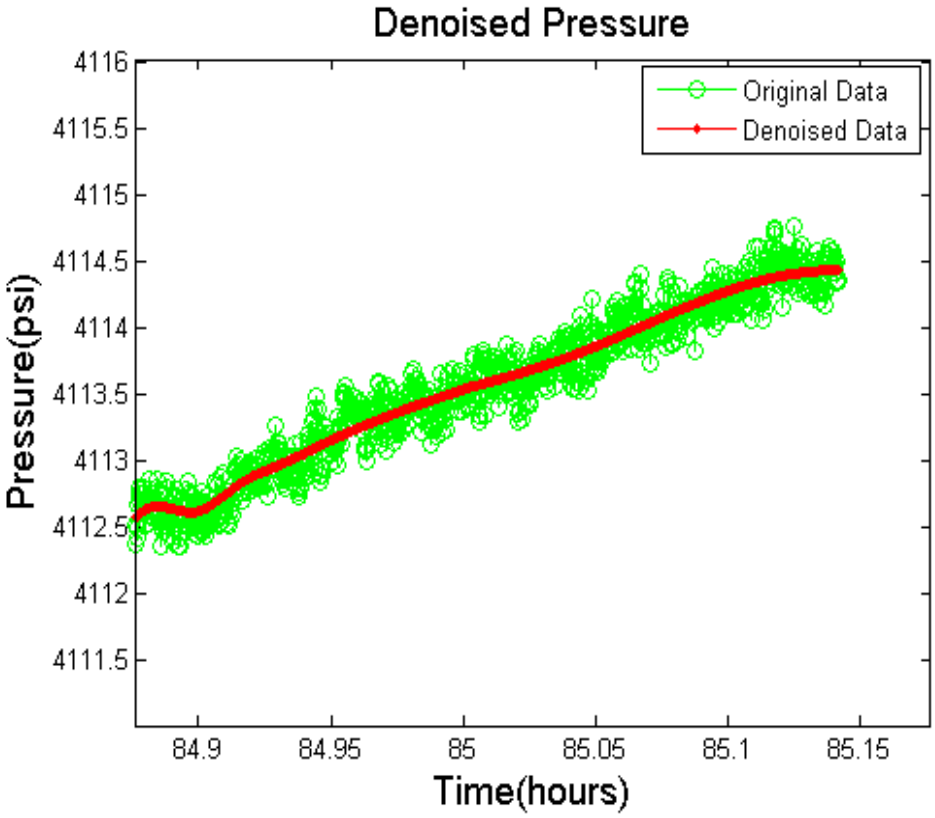


Figure 2.66 Zoom-in of denoised BU of Field Example 2

2.8.2 Data reduction

The size of data sets acquired with permanent pressure gauges is enormous. A gauge system with a 10-second recording interval registers more than three million data points a year. It is cumbersome even to plot the data to see the general behavior, not to mention analyzing them. Therefore, once the data denoising is complete, one may choose to reduce the data to more manageable levels for interpretation with close resemblance to original data and details of information contained within. As the data interpretation has not yet been carried out at this stage it is imperative that numerical and or biased artifacts do not significantly influence the data. Data reduction is a type of data sampling. It follows that the data reduction must obey the Shannon theory. We mainly carry out data reduction in two different ways. One is to perform data reduction for the whole dataset for reservoir simulation and data observation. The other is to perform data reduction for single flow period analysis.

One of the methods for reducing the number of data is the pressure threshold method. In this method, the data are sampled only when a certain change in pressure is reached. However, using a threshold on pressure alone is not sufficient, as the pressure may stay

relatively constant over long periods of time. The pressure threshold criterion may produce large gaps in data. A time limit should thus be imposed on the sampling space. The pressure should be recorded when the change in pressure is higher than a maximum preset pressure value (ΔP_{max}) and whenever the time span between samples becomes higher than a maximum preset time threshold (Δt_{max}). Figure 2.67 is the original dataset which consists of 0.11 million data from H oil field. The H field is characterised by basinward-verging anticlines and associated thrust faults. The field is being developed by 12 wells produced with a single-piece truss spar permanently moored in 4,500ft water depths. After data reduction, the compressed dataset shows the same trend and fluctuation but contains only 5649 data points.

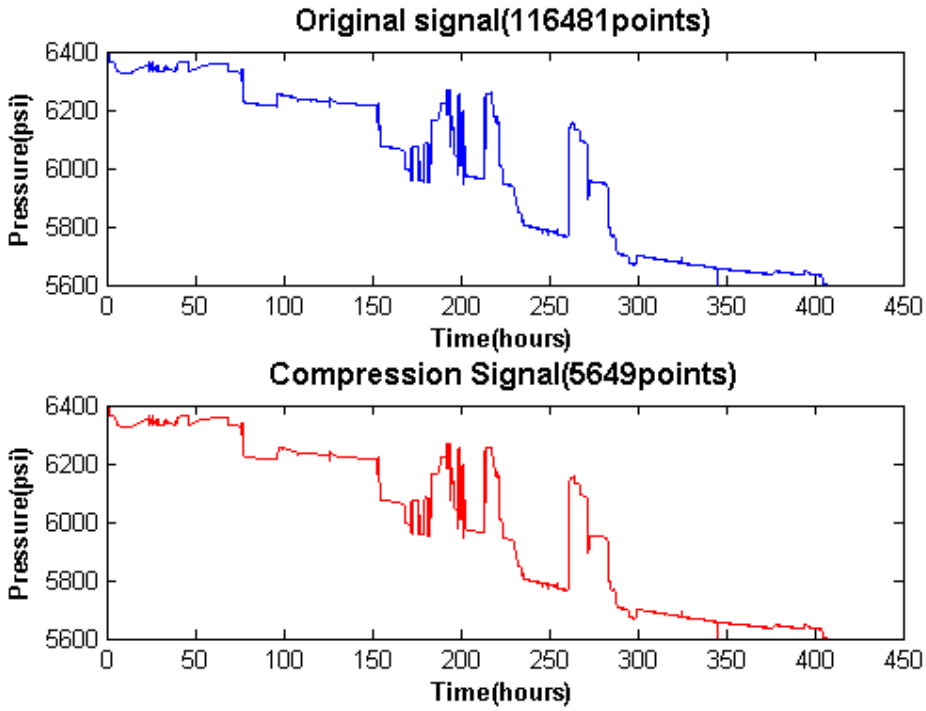


Figure 2.67 data reduction of the D field dataset

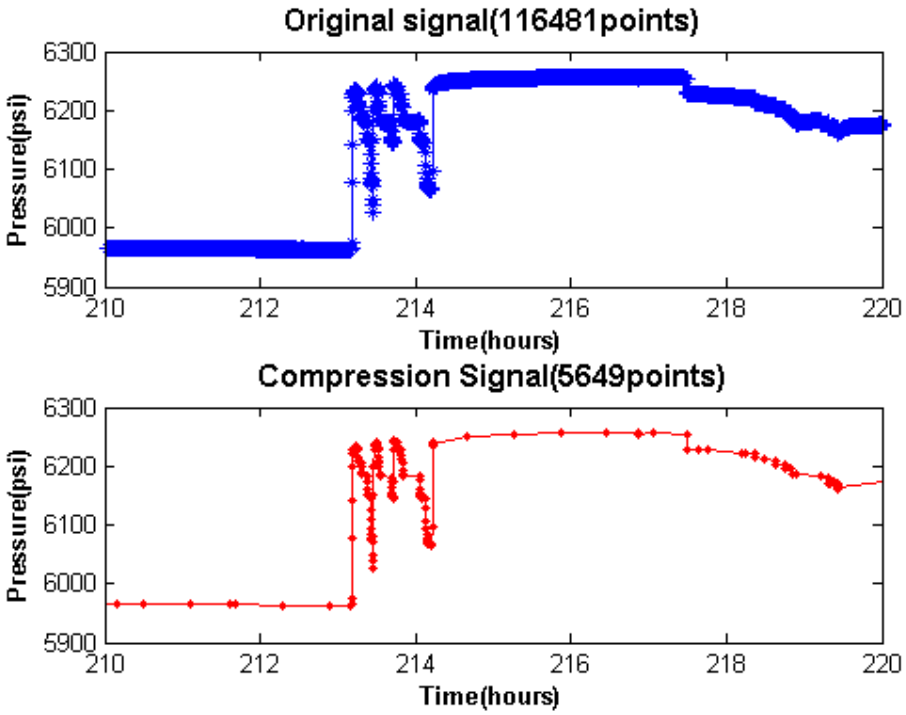


Figure 2.68 Zoom-in of the data reduction of the D field dataset

Figure 2.68 shows a part of the whole dataset. We can observe the lower frequency data was represented with few data points whereas the sharply changed part is represented with more data points.

The second application is to perform data reduction for a single flow period before analyzing it. The derivative is still dramatically changed although the pressure curve is very smooth in the Cartesian plot. Unfortunately, the higher the frequency we use to record data, the more noise is added into the signal. Therefore the key to processing these data is to sample the data at unequal time intervals. A small interval is selected when the signal changes faster, while the larger interval is used when the signal changes slowly. On the other hand, a few points are enough for a log-log plot to identify the straight line.

The following algorithm is applied to obtain equal time intervals in the log scale. This relationship can be written as the following equation: $\log(t) = an + b$ Where t is the time of point n and a,b is the coefficient which depends on the total time of the flow period, the minimum time scale and the total number of data points. Suppose the total time of one flow period is t_{total}. The minimum time scale is $t_{\min}$. The minimum time scale can be set as the time interval for recording PDG pressure and N is the total number of data points which can represent this flow period. So we can obtain the

following equation:

$$\begin{cases} \log(t_{\min}) = a*1+b \\ \log(t_{total}) = a*N+b \end{cases} \tag{2-12}$$

Solving the upper equation array; we can obtain the proper equation:

$$\log(t) = \frac{n-1}{N}(\log(t_{total}/t_{\min})) + \log t_{\min} \tag{2-13}$$

Figure 2.69 shows a dataset of field example 2 in which the original signal is 511702 points and the compression coefficient is 100. From the graph, the reduced signal can express the 0riginal signal. The procedure of reduction consists of two parts. Retain all high frequency points at the beginning period. This initial information is every important for well testing analysis. The low frequency points will be sampled by increased sample steps.

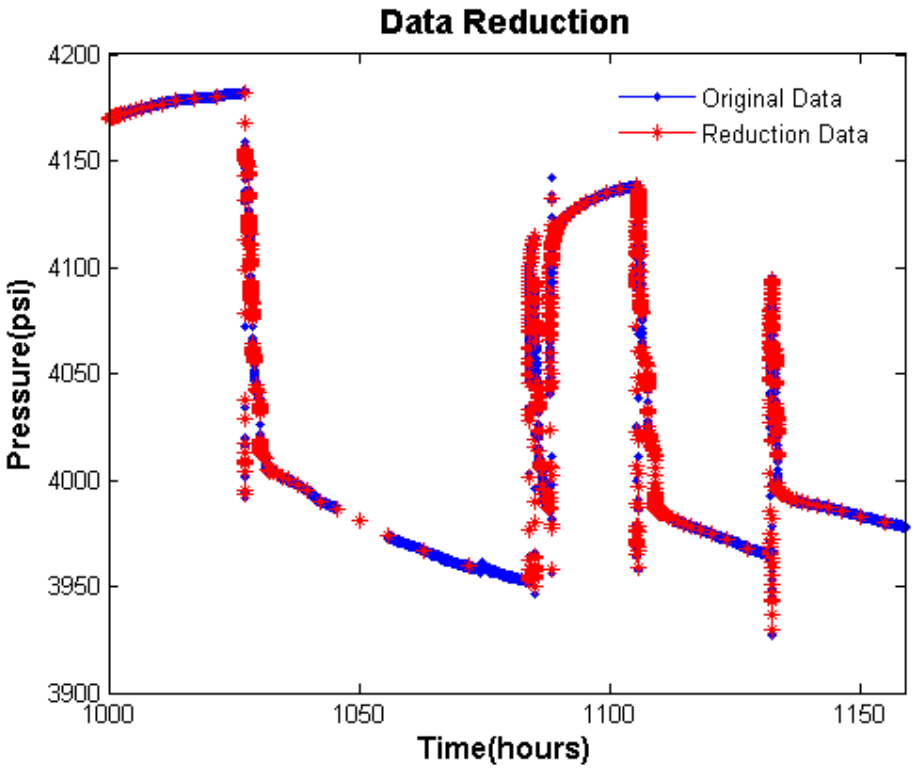


Figure 2.69 Data reduction of Field Example 2

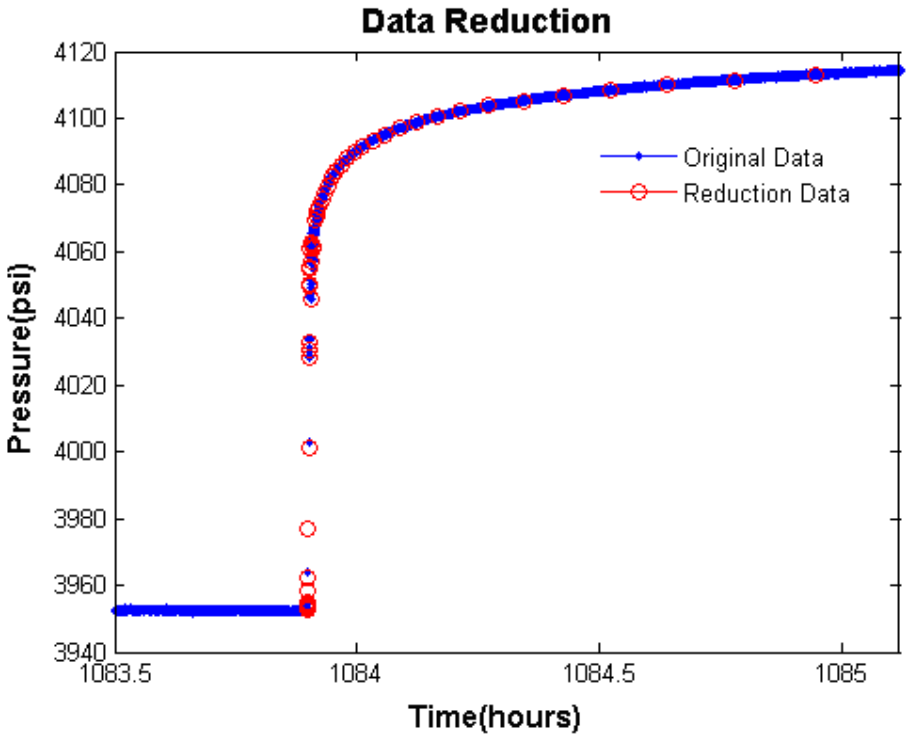


Figure 2.70 Zoom-In of Data reduction Field Example 2

Figure 2.70 shows the result after this unequal interval sampling. It can be observed that the data points are very dense at the beginning of the last BU. The later time data points are sparser.

2.9 Chapter conclusion

A new workflow based on wavelet approach has been presented to process PDG pressure data. In this workflow, the PDG pressure data is treated as an integrated signal from the gauge, well and reservoir. The procedure of data processing filters different information in several steps: data preprocessing, outlier removal, flow event detection, identification of BU and DD, data denoising and data reduction.

This workflow first handles the effects of the gauge and acquisition system by data preprocessing and outlier removal. The data preprocessing is to remove unreasonable data information, such as negative and repeated values. The outlier removal step can remove spike and step outliers caused by the gauge. After filtering the gauge information, the well information is detected. The key step is to identify flow events. It was found the variation of pressure data is greater when the flow events happen. These phenomena can be detected in the detail signal after the wavelet transform. A new algorithm has been designed to achieve this and tested using real PDG data.

After flow event identification, the further study is to mine the reservoir and well information of each flow period. With the help of temperature information and production information, this step accomplishes the distinguishing of flow event into the DD, shut-in BU and rate-drop BU. In addition, this study also further identifies some special phenomena like the multi-DD, fluctuation and well interference. However, there are still some special phenomena which we can not understand.

The last step is to perform denoising and data reduction. It is suggested that denoising part of the data is better than denoising the whole data because this can avoid smearing effects. The data reduction stage suggests two ways to reduce data. If the user wants to analyze the data with the well testing theory, the data reduction can sample data with a log scale.

CHAPTER 3 PDG PRESSURE DATA ANALYSIS

3.1 Introduction

After the PDG data processing, the PDG pressure data are separated into different BU and DD. These series of BU and DD can be analyzed from the view of well testing. Well testing analysis has been developed for more than fifty years. The trandtional well testing approach like straight line method and log-log derivative was derived from the analytical solution of diffusivity equation. Recently, the numerical approach called numerical well testing has been accepted. This chapter will introduce how to analyze the PDG pressure with these approaches.

Before data analysis, accurate production history is very important issues because the real rate of production is far from the actual production history. Sometimes, there is only the month cumulative production or daily rate. Therefore, this chapter first proposes a new technique of integrating PDG pressure and accumulated production rate to recover the production history.

Having obtained the accurate PDG pressure and production rate, we will first consider how the PDG pressure data is analyzed with the traditional approach. A new flow is designed to analyze the PDG pressure with traditional approach. All the BU are selected from PDG data. The longest BU is analyzed by the reservoir engineer and, based on the engineer's experience, is saved as a template for all the remaining BU. The same model and workflow will be applied to the remaining BU to obtain the changed parameters of reservoir, such as skin factor, permeability and boundary.

The traditional approach to analysis focuses on BU and neglects the effect for DD. The analysis model is also very simple compared with the real complex and heterogeneous reservoir. Therefore, we will consider analyzing the PDG data with numerical well testing which integrates geological information and reservoir dynamic information. A new toolbox will be designed to relize the automatical numerical well testing by calling the commercial simulator.

3.2 Flow rate recovery

The PDG pressure data have been widely installed to record the pressure and temperature in real time. However, few companies install the downhole flow meter, for economic reasons. Some oil companies measure the production rate on the surface every day or every week. This low frequency production history data can not represent the real downhole production rate. So, it is a major problem to match the real PDG pressure profile with this low frequency production history. This is worsened further if only production history is recorded for a grouped well on the platform. The inaccurate production history will increase the uncertainty of reservoir simulation and result in mistakes in making decisions. Therefore, flow history reconstruction is extremely important for PDG data anlaysis.

The new approach presented in this study will recover the rate from the view of system and signal. The key of reallocating the rate is to know the relation between the rate and real time pressure. This relation is a complex and nonlinear equation which cannot be described by current formulae. When the wavelet transform is used to process the PDG pressure, more attention is paid to processing data in high frequency signal to identify the flow event. But no attention is paid to the relation between the amplitude of high frequency signal and rate. This relation is found to be a linear relation which can be used to recover the rate under the restriction of accumulative production. In next section, the theory of this relationship will be described while the procedure of rate recovery will be discussed in the case study section.

3.2.1 Theory of flow rate recovery

The reservoir is a continuous-time dynamic system in which continuous-time input signals result in continuous-time output signals. The production rate is considered as the input signal, while the pressure is output signal caused by the reservoir system. The DD and BU can be considered as step signals. Every short term rate variation acts like an impulse signal for reservoir system. For a linear system, the response of a unit impulse signal is constant. This means the proportion between rate variations and pressure response is a constant. This relation can be used to calculate approximately the rate from PDG pressure under the restriction of accumulated production. Although there is no real linear system in real world, a nonlinear system can be treated as a linear system over a short time. Figure 3.1 illustrated these reservoir system and signal.

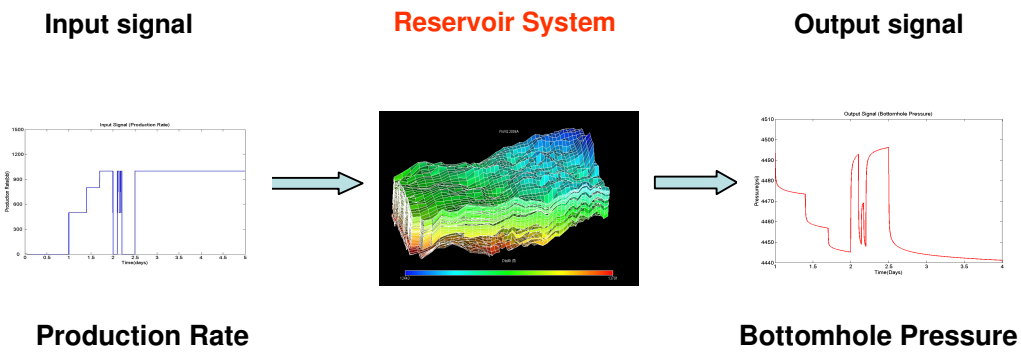


Figure 3.1Reservoir system and signal

The key of applying this theory is to find the step signal and Linear Time Invariant (LTI) system. Any real physical system has some inertia associated with it and thus does not respond instantaneously to inputs. Consequently, if a step of sufficiently short duration is applied to such a system, the system response will not be noticeably influenced by the step's duration or by the details of the shape of the step signal. For any physical system, we can always find a pulse that is 'short enough'. In our reservoir system, any variation of production rates in instantaneous time can be considered as an approximate step signal.

In the other hand, there is no real LTI system in real world because the parameters of the system are all changing during its life. But a nonlinear-system can be considered as a linear system over a short time. This type approximation also enables description of the the real world. The next part will show the relation between rate and pressure in a single phase oil reservoir system.

Let us begin from a single phase oil reservoir model. The differential equation for fluid flow in a porous medium, the diffusivity equation, is a combination of the law of conservation of matter, an equation of state, and Darcy's law.

$$\frac{\partial^2 p}{\partial r^2} + \frac{1}{r}\frac{\partial p}{\partial r} = \frac{\phi\mu c_t}{k}\frac{\partial p}{\partial t} \tag{3-1}$$

This equation assumes horizontal flow, negligible gravity effects, a homogeneous and isotropic porous medium, a single fluid of small and constant compressibility, and applicability of Darcy's law, and that ϕ, μ, c_t, k are independent of pressure. The linearity of the diffusivity equation allows the application of the super-position theorem as a sequence of constant terminal pressures or constant rates in such a fashion that it reproduces the pressure or production history at the boundary. In the assumption, the

ϕ, μ, c_t, k are constant allowing the conclusion that this system is a LTI system.

The following equation is the solution for Infinite Radial flow Reservoir of equation 3-1:

$$p(r,t) = p_i - \frac{q\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r^2}{4kt})\} \tag{3-2}$$

$$p_{wf} = p_i - \frac{q\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4kt})\} \tag{3-3}$$

Where p_i is the initial pressure; p_{wf} is the Bottomhole pressure; $-Ei(-x) = \int_x^{\infty} \frac{e^{-u}}{u} du$;

Consider the simplest situation of the variation of production rate. Suppose the production history includes two periods. First, the well production rate is q_1 for time t_1 then the well production rate is q_2 for Δt. Then,

$$p_{wf1} = p_i - \frac{q_1\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4kt_1})\}$$

$$\Rightarrow p_i - p_{wf1} = \frac{q_1\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4kt_1})\} \tag{3-4}$$

$$p_{wf2} = p_i - \frac{q_1\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4k(t_1+\Delta t)})\} - \frac{(q_2-q_1)\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4k\Delta t})\}$$

$$\Rightarrow p_i - p_{wf2} = \frac{q_1\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4k(t_1+\Delta t)})\} + \frac{(q_2-q_1)\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4k\Delta t})\} \tag{3-5}$$

Where, P_{wf1} is the downhole pressure when the well produce q_1 for t_1 hours; P_{wf2} is the downhole pressoure when the well produce q_2 for another Δt hours.

Compare the two pressure with (3-5) - (3-4) :

$$p_{wf1} - p_{wf2} = \frac{q_1\mu}{2\pi kh}\{\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4k(t_1+\Delta t)})\} - \{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4kt_1})\}\}$$
$$+ \frac{(q_2-q_1)\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{\phi\mu c r_w^{\ 2}}{4k\Delta t})\} \tag{3-6}$$

Ask:

$$t_D = \frac{0.0002637kt}{\phi\mu c_t r_w^{\ 2}} \qquad r_D = \frac{r}{r_w} \qquad \Delta p = p_{wf1} - p_{wf2}$$

Then

$$\Delta p = \frac{\mu}{2\pi kh}\left\{ q_1\{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4(t_{D1}+\Delta t_D)}) - \{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4t_{D1}})\}\} + (q_2-q_1)\{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4\Delta t_D})\}\right\} \tag{3-7}$$

From the equation 3-7, we can find Δp includes two parts. The first parts is affected by the q_1 for Δt time. The second part is the effect from the changed rate. If the Δt is much smaller than t_1, the first part can be omitted. Equation 3-7 transfers into equation 3-8.

$$\frac{\Delta p}{\Delta q} \approx \frac{\mu}{2\pi kh}\{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4\Delta t_D})\} \tag{3-8}$$

From equation 3-8, we can found the ratio between pressure and rate is constant.

As we proposed, the unit impulse response of LTI system should be a constant. So, the front part of equation must be omitted in order to obtain the true response to impulse signal. The next part will discuss the condition in which the signal can be considered as the impulse signal. So, we need to compare the value of $q_1\{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4(t_{D1}+\Delta t_D)}) - \{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4t_{D1}})\}\}$ and $(q_2-q_1)\{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4\Delta t_D})\}$.

Suppose:

$$C = \frac{(q_2-q_1)}{q_1} * \frac{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4\Delta t_D})}{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4(t_{D1}+\Delta t_D)}) - \{-\frac{1}{2}Ei(-\frac{r_D{}^2}{4t_{D1}})\}} \tag{3-9}$$

$$M = \{-\frac{1}{2}Ei(-\frac{1}{4\Delta t_D})\} / \{-\frac{1}{2}Ei(-\frac{1}{4(t_{D1}+\Delta t_D)})\} - \{-\frac{1}{2}Ei(-\frac{1}{4t_{D1}})\} \tag{3-10}$$

$$N = [(q_2-q_1)/(q_1)]$$

If the C is bigger enough, the first part of equation 3-7 can be neglect. The reservoir system can be considered as a linear system.

The following example will help to prove this. There is an example from book 'Advances in well test analyses'. This example asks reader to estimate the pressure at a well located in the centre of a closed-square reservoir after it has produced 135STB/D of dry oil for 15 days. Other data are

$p_i = 3265\ psi$ $r_w = 0.50\ ft$

$\phi = 0.17$ $B_0 = 1.02$

$k_0 = 90 md$ $A = 40 acres$

$\mu_0 = 13.2 cp$ $h = 47\ ft$

$c_t = 2.00 * 10^{-5}\ psi^{-1}$ $s = 0$

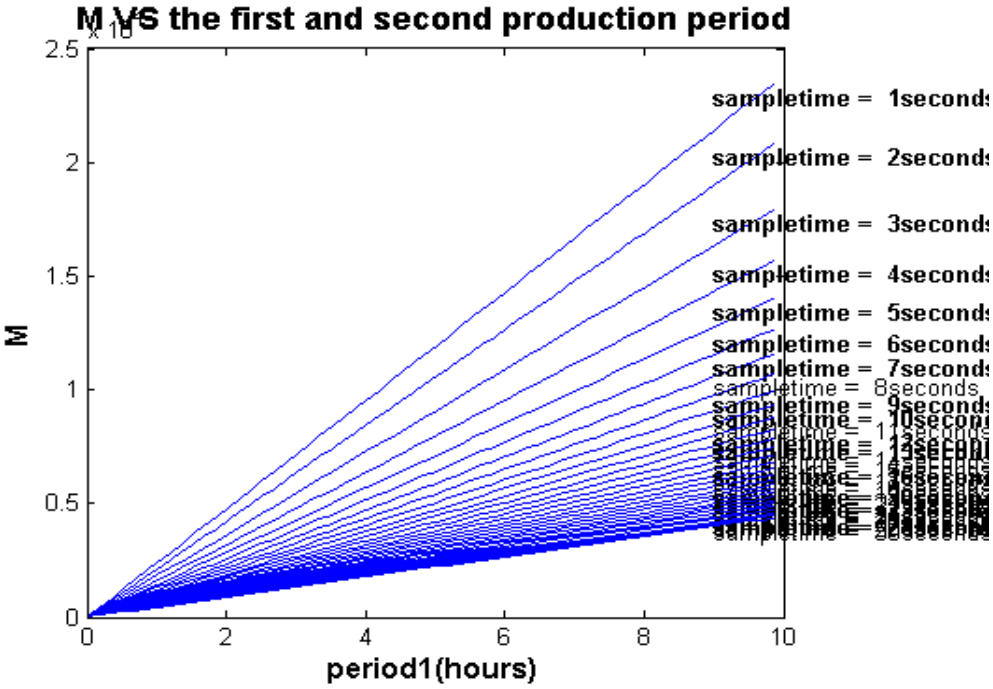


Figure 3.2 C value with constant N in oil reservoir

Figure 3.2 shows the M change with t_{D1} and Δt_D while N is equal to 1. The x axis is the production time of q_1 and the Y axis is the C over a short time. The sample time is the production time of q_2. From the graph, if the t_{D1} is one hours and Δt_D is one second, C is equal to 2500. This means the second part of equation 3-7 is 2500 times of the first part of equation 3-7. So, the first part can be omitted. Equation 3-7 can be written as equation 3-8. And the ratio between pressure and rate is a constant value.

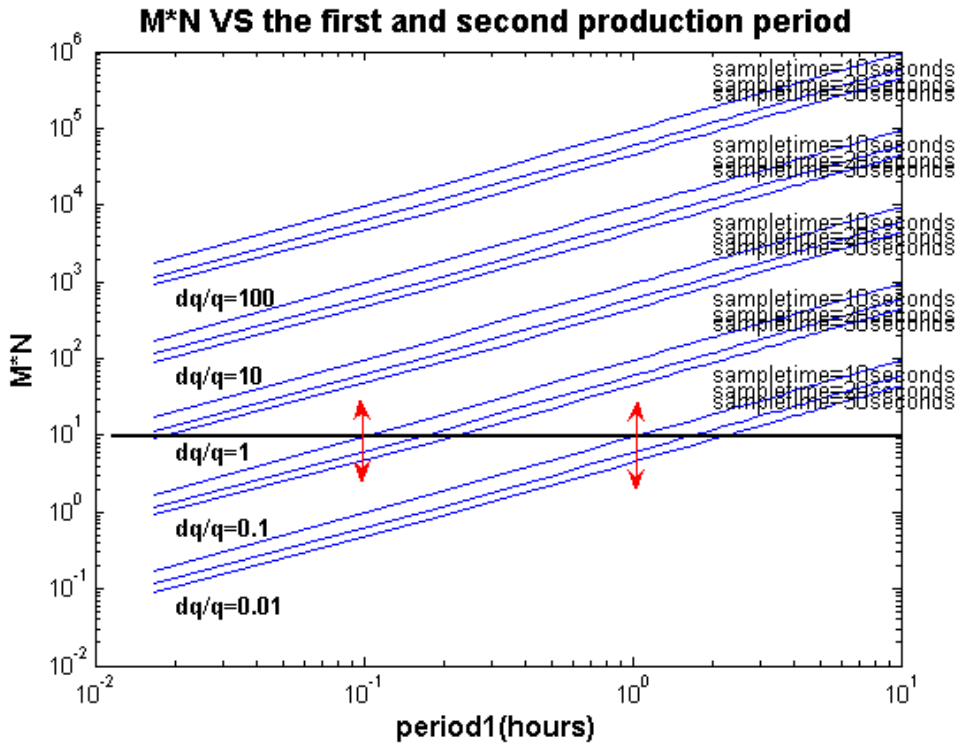


Figure 3.3 C value with changed N in oil reservoir

From Figure 3.3, we know that if the range of rate change is smaller than 10%, t_{D1} must maintained for more than 1 hour and Δt_D is less than 10 seconds to obtain the impulse response. But this small rate change is always the noise. From this figure, the condition of the impulse signal for the reservoir system can be found. It can be seen that the results validate the relation between the rate variation and the response of the reservoir is linear.

The next part shows how to decide the step signal in time domain. It is very hard to find the response to a unit step signal from a long term signal. The step signal can be considered as a type of high frequency signal at the point where the input changes. So, we can use a high frequency filter to obtain this type of signal. The wavelet transform is a good tool to obtain the step signal.

The CWT is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function:

$$C(scale, position) = \int_{-\infty}^{\infty} f(\tau)\psi(scale, position, \tau - t)d\tau \quad (3\text{-}11)$$

$$C(scale, position) = f(t) * \psi(scale, position, t) \quad (3\text{-}12)$$

Where: $f(t)$ is the original signal $\psi(scale, position, t)$ is the wavelet

function; $C(scale, position)$ is the frequency signal of original signal.

From equation 3-11, since $f(t) = \int_{-\infty}^{\infty} Au(t)h(t-\tau)dt = Ah(t)$ then,

$$C(scale, position) = Ah(t) * \psi(scale, position, t) \quad (3\text{-}13)$$

$$C(scale, position) = Au(t) * \psi(scale, position, t) * system(t) \quad (3\text{-}14)$$

From equation 3-14, the result of a step signal response is still a constant after it is transformed from wavelet transform. This results in the amplitude of high frequency wavelet signal having a linear relationship to the input step signal.

According to the theory of signal and system, the proportion between the variation of production rate and the amplitude of high frequency pressure is constant at the point where the production rate changes. The event detection by wavelet transform can help to distinguish the production period and the response of variation of rate. Then the production history can be recovered from the pressure together with the accumulated production rate.

The procedure for applying the approach presented in this study is quite straightforward. As long as the PDG data and accumulated data are applied, the following four-step procedure can be applied for recovering the rate. The first step is to process the pressure data. Outlier removal and denoising is applied to obtain the true signal of the reservoir. Then the data is interpolated into equal step data. The following step is to detect the flow event. Wavelet transform can help to distinguish the BU and DD. The third step is to identify the response of reservoir to the variation of production rate. This step is to find the maximum amplitude of the high frequency pressure signal at the beginning and end of BU and DD. The fourth step is to recover the production history from pressure together with the accumulated production.

3.2.2 Case Studies

I. Case Study 1 ----Recovering rate from PDG pressure and cumulative production

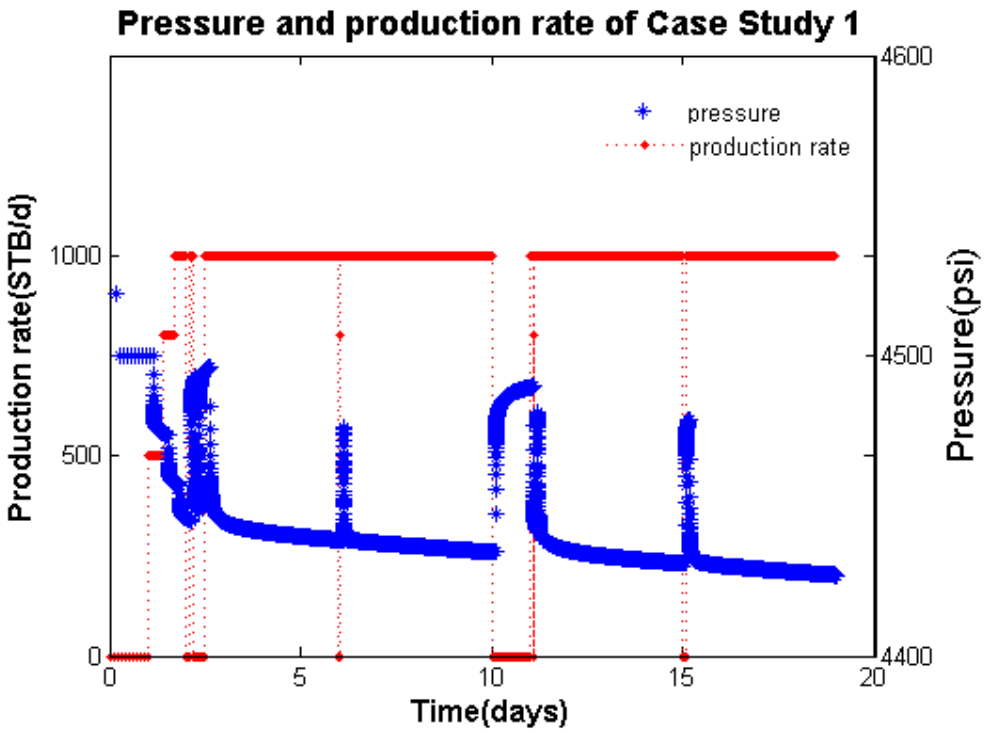


Figure 3.4 the PDG pressure and production history of Case Study 1

This case builds a homogeneous and single phase oil reservoir model using ECLIPSE . The production history is showed in figure 3.4. The starred line is pressure and the dotted line is the production rate of reservoir. Table 3.1 gives the details of the production history. The production history begins at a one day shut-in and followed by multi-rate production and a long-term BU. Then the normal production is operated. This case simulates a period of more than 18 days and a time interval of 0.001 days. In this case, the wellbore storage is not considered and the skin factor is 3.

Beg(Day)	End(Day)	Time-step(Day)	Rate(STB/D)
0	1	0.1	0
1	1.4	0.001	500
1.4	1.7	0.001	800
1.7	2	0.001	1000
2	2.1	0.001	0
2.1	2.14	0.001	1000
2.14	2.17	0.001	500
2.17	2.2	0.001	1000
2.2	2.5	0.001	0
2.5	6	0.001	1000
6	6.01	0.001	0
6.01	6.02	0.001	500
6.02	6.03	0.001	800
6.03	11.1	0.001	1000
11.1	11.11	0.001	0
11.11	11.12	0.001	800
11.12	11.13	0.001	0
11.13	15	0.001	1000
15	15.1	0.001	0
15.1	19	0.001	1000

Table 3.1 Production history of Case Study 1

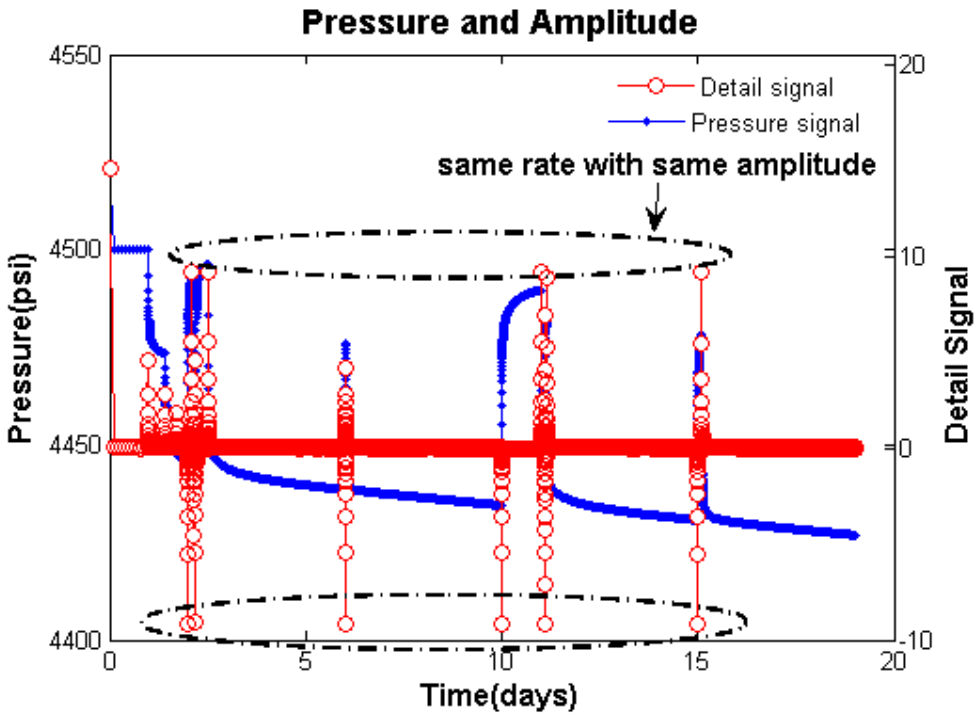


Figure 3.5 Pressure and Detail signal of Case Study 1

The fist step for rate recovery is to process the PDG data with the wavelet transform in order to identify the flow events and to identify the BU and DD. Figure 3.5 shows the

PDG pressure and the detail signal after wavelet transform. The doted line in graph 3.5 is pressure and the circled line is the high frequency signal of pressure after wavelet transform. We can find that the beginning of a DD corresponding with positive amplitude and the beginning of BU corresponds with negative amplitude. This principle can be used to distinguish the BU and DD but can not distinguish the shut-in BU from the rate-drop BU because these two types of BU both exhibit negative amplitude. At the same times, we also find the amplitude at the beginning of every flow period is approaximatly same value when the rate variation is same.

Beg	end	Period(day)	Rate	Amplitude	Sum	adjust
0	1	1	0			
1	1.4	0.4	500	4.61	4.61	4.61
1.4	1.7	0.3	800	2.76	7.37	7.37
1.7	2	0.3	1000	1.84	9.20	9.20
2	2.1	0.1	0	-9.15	0.06	0.00
2.1	2.14	0.04	1000	9.18	9.18	9.18
2.14	2.17	0.03	500	-4.52	4.65	4.65
2.17	2.2	0.03	1000	4.56	9.22	9.22
2.2	2.5	0.3	0	-9.11	0.10	0.00
2.5	6	3.5	1000	9.20	9.20	9.20
6	6.01	0.01	0	-9.14	0.06	0.00
6.01	6.02	0.01	500	4.15	4.15	4.15
6.02	6.03	0.01	800	2.79	6.94	6.94
6.03	10	3.97	1000	1.95	8.89	8.89
10	11	1	0	-9.14	-0.24	0.00
11	11.1	0.1	1000	9.19	9.19	9.19
11.1	11.11	0.01	0	-9.12	0.07	0.00
11.11	11.12	0.01	800	6.92	6.92	6.92
11.12	11.13	0.01	0	-7.13	-0.21	0.00
11.13	15	3.87	1000	8.88	8.88	8.88
15	15.1	0.1	0	-9.13	-0.25	0.00
15.1	19	3.9	1000	9.15	9.15	9.15

Table 3.2 Amplitude of detail signal in Case Study 1

Table 3.2 shows the amplitude of the detail signal after wavelet transform. The Sum column is the sums of the amplitude from the beginning to the end. A principle can be found that the amplitude is almost zero when the BU is a shut-in BU. This phenomenon can be applied to detect the shut-in BU. So, if the absolute summed value of amplitude is smaller than 5 percent of the neighborhood absolute value, this period will be considered as a shut-in BU.

After the identification of BU and DD, it has been detected where the rate changed during the whole PDG pressure. The theory of the previous section concluded that the variation of production rate is linear to the amplitude of pressure. Graph 3.6 shows

that variation of production rate accompanies the changed amplitude. The production rate of the first three flow period is 500stb/day, 800stb/day and 1000stb/day. The variation of every flow period is 500stb/d, 300stb/d and 200stb/day. The amplitude is 4.61, 2.76 and 1.84. The linear relationship can be deduced from these three values. Table 3.3 presented the detail information of every flow period.

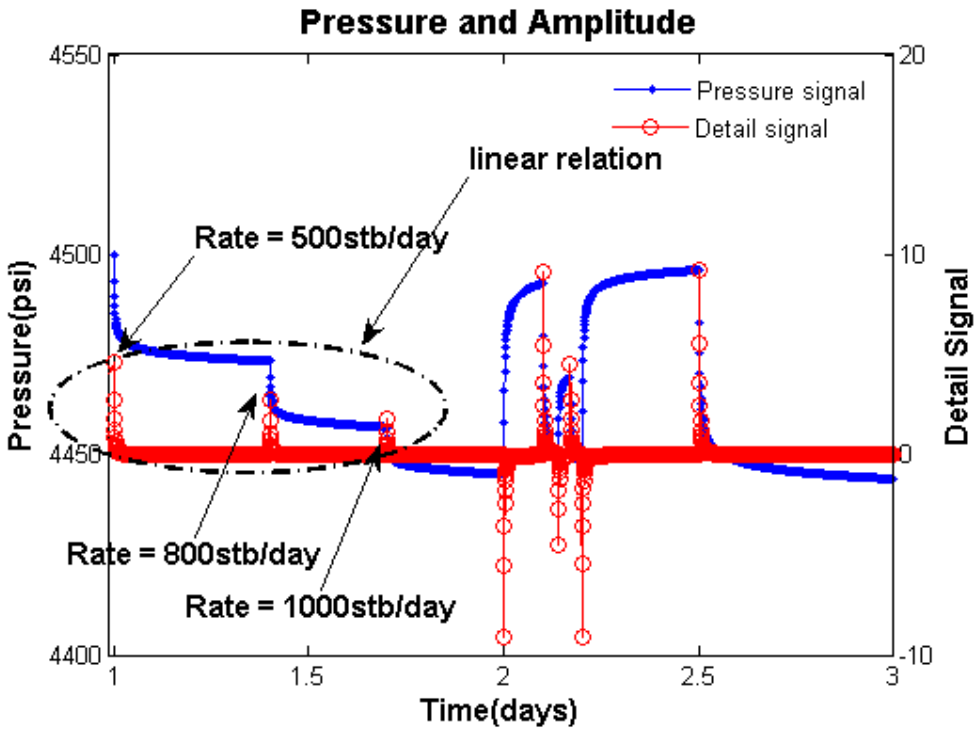


Figure 3.6 Linear relationship between amplitude and rate variation in Case study 1

From Table 3-3, it can be seen in the last column that the amplitude divided by production rate is approximately constant. This result confirms the theory which indicates that the proportion between variation of rate and the amplitude of high frequency pressure is constant.

Beg(day)	end(day)	Rate	Amplitude	Rate/Amplitude
0	1	0		
1	1.4	500	4.61	0.0092
1.4	1.7	800	2.76	0.0092
1.7	2	1000	1.84	0.0092
2	2.1	0	-9.15	0.0091
2.1	2.14	1000	9.18	0.0092
2.14	2.17	500	-4.52	0.0090
2.17	2.2	1000	4.56	0.0091
2.2	2.5	0	-9.11	0.0091
2.5	6	1000	9.20	0.0092
6	6.01	0	-9.14	0.0091
6.01	6.02	500	4.15	0.0083
6.02	6.03	800	2.79	0.0093
6.03	10	1000	1.95	0.0092
10	11	0	-9.14	0.0091
11	11.1	1000	9.19	0.0092
11.1	11.11	0	-9.12	0.0091
11.11	11.12	800	6.92	0.0086
11.12	11.13	0	-7.13	0.0089
11.13	15	1000	8.88	0.0089
15	15.1	0	-9.13	0.0091
15.1	19	1000	9.15	0.0091

Table 3.3 Relationship between amplitude and production rate in Case Study 1

After the identification of BU and DD and the establishment of the linear relationship, the following algorithm is developed to recover rate. Suppose the production history has n period and the accumulated production rate is R_{sum}, and the amplitude of every period is a_j. The response of unit impulse is a constant, h, so we have the equation between the production rate and amplitude.

$$R_{sum} = \sum_{i=1}^{n} t_i \sum_{j=1}^{i} a_j h \tag{3-15}$$

So the production rate of every period is R_i:

$$R_i = a_j h = a_j * \frac{R_{sum}}{\sum_{i=1}^{n} t_i \sum_{j=1}^{i} a_j} \tag{3-16}$$

Table 3.4 shows how to recover the detail of production history. The error of estimated rate is in the range of 5%, and the error increases as the production period decreases . Figure 3.7 shows the error between the real rate and recovered rate.

Period (day)	rate (bbl/d)	amplitude	Sum	adjust	Recovered Rate	error
1	0					
0.4	500	4.61	4.61	4.61	509.73837	1.9
0.3	800	2.76	7.37	7.37	815.15423	1.8
0.3	1000	1.84	9.2	9.2	1018.5361	1.8
0.1	0	-9.15	0.06	0	0	0
0.04	1000	9.18	9.18	9.18	1015.4042	1.5
0.03	500	-4.52	4.65	4.65	514.81794	2.9
0.03	1000	4.56	9.22	9.22	1019.8641	1.9
0.3	0	-9.11	0.1	0	0	0
3.5	1000	9.2	9.2	9.2	1018.3812	1.8
0.01	0	-9.14	0.06	0	0	0
0.01	500	4.15	4.15	4.15	470.95378	4.2
0.01	800	2.79	6.94	6.94	768.20974	3.9
3.97	1000	1.95	8.89	8.89	984.18538	1.5
1	0	-9.14	-0.24	0	0	0
0.1	1000	9.19	9.19	9.19	1017.0532	1.7
0.01	0	-9.12	0.07	0	0	0
0.01	800	6.92	6.92	6.92	765.69762	4.2
0.01	0	-7.13	-0.21	0	0	0
3.87	1000	8.88	8.88	8.88	983.01232	1.6
0.1	0	-9.13	-0.25	0	0	0
3.9	1000	9.15	9.15	9.15	1012.3499	1.2

Tables 3.4 Recovering the production rate in Case Study 1

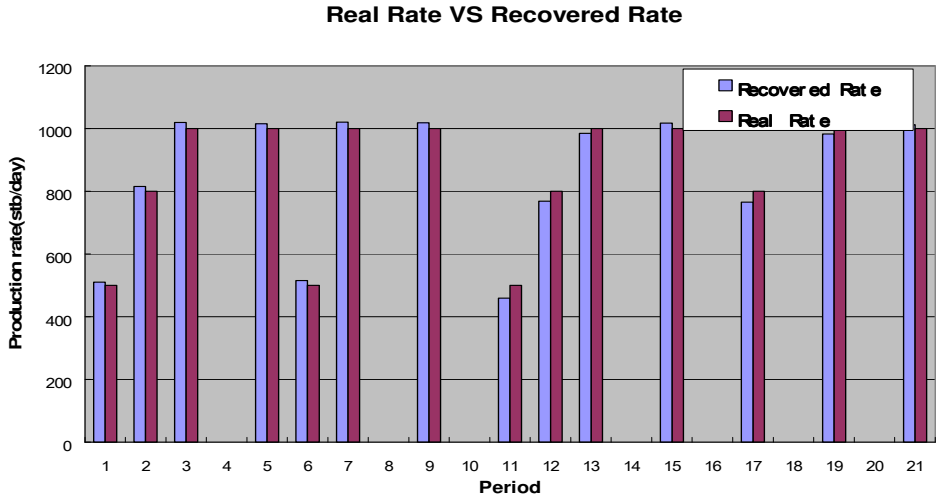


Figure 3.7 Real rate vs recovered rate in Case Study 1

II. Case Study 2 ----Recovering rate from PDG pressure and daily production

The Case Study 1 introduced the basic algorithm of rate recovery from PDG pressure and accumulated rate. The accumulated rate in Case Study 1 is one total production value in a period. However, the normal production history in the oil company is the

daily production rate. So, this case will investigate how to recover rate history from daily production rate and PDG pressure. Case study 2 is single phase oil sandstone reservoir. The permeability of reservoir is 1 Darcy. One well start to produce without wellbore storeage. The skin factor of well is 3The downhole meter is installed to record the rate. Figure 3.8 shows the PDG pressure and daily rate of Case Study 2. As the graph shows, there is a shut_in period during the second day but this can not be found in the daily rate. So, the daily rate can not reflect the real production history because of the low frequency of the production rate. It's necessary to recover the rate from PDG pressure and daily rate.

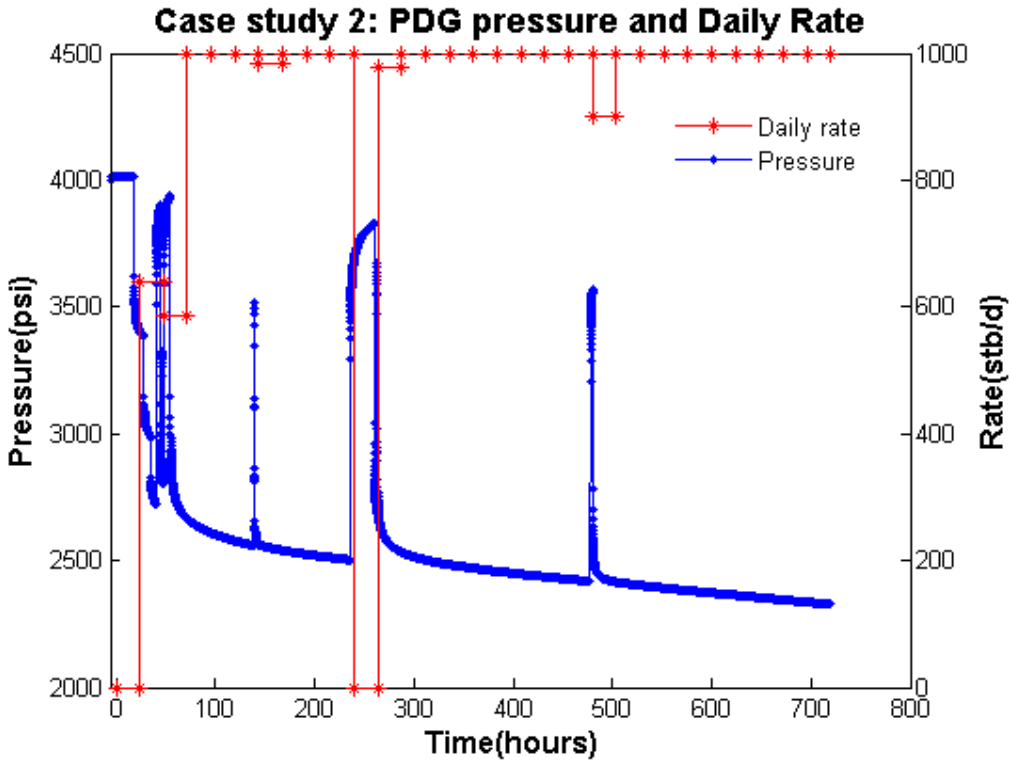


Figure 3.8 Daily rate and PDG pressure

The follow step is first to perform wavelet transform to obtain the high frequency signal of PDG pressure. Figure 3.9 show the detail signal and the original PDG pressure signal. As we found, there is a peak when the rate changed.

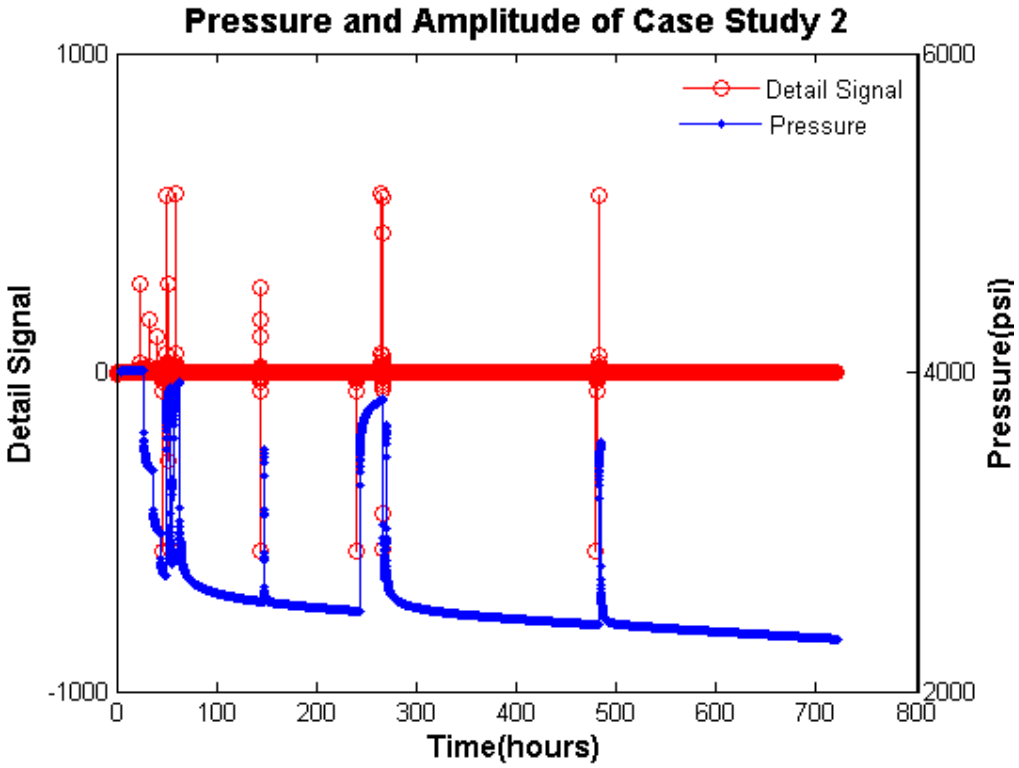


Figure 3.9 Detail signal and PDG pressure

Number	Flag	Beginning	End	Period	Amplitude	Accumulated
1	1	24	33.6	9.6	278.9	278.9
2	1	33.6	40.8	7.2	167.54	446.44
3	1	40.8	45.6	4.8	111.84	558.28
4	-1	45.6	50.4	4.8	-558.05	0.22564
5	1	50.4	51.36	0.96001	557.49	557.72
6	-2	51.36	52.08	0.72	-276.55	281.17
7	1	52.08	52.8	0.72	278.48	559.65
8	-1	52.8	60	7.2	-556.3	3.3426
9	1	60	144	84	557.71	561.06
10	-1	144	144.24	0.24001	-558.34	2.715
11	1	144.24	144.48	0.23999	267.78	270.5
12	1	144.48	144.72	0.24001	167.55	438.05
13	1	144.72	240	95.28	114.18	552.22
14	-1	240	264	24	-558.28	-6.0598
15	1	264	266.4	2.4	557.85	551.79
16	-1	266.4	266.64	0.23998	-557.3	-5.5106
17	1	266.64	266.88	0.24001	435.97	430.46

Tables 3.5 Amplitude of each period

The initial steps for recovering rate are all same as with Case Study 1, but the following step is different from it because the daily rate is known and some periods are longer than 24 hours. In tables 3.5,when the flow period is longer than 24 hours, the production rate can be derived from daily rate. If the flow period is smaller than 24 hours, this means there is more than one flow event during this day. The rate history

can be recovered with the PDG pressure and this one day production rate. The rate at beginning and end of this individual period also need be corrected with the rate of the nearest period.

Begin(h)	End(h)	Recovered Rate(stb/d)	Real Rate(stb/d)	Error
0	24	0	0	0
24	33.6	499.7816031	500	0.043679
33.6	40.8	800.0089599	800	-0.00112
40.8	45.6	1000.423354	1000	-0.04234
45.6	48	0	0	0
48	50.4	0	0	0
50.4	51.36	997.3239869	1000	0.267601
51.36	52.08	502.7927731	500	-0.55855
52.08	52.8	1000.775244	1000	-0.07752
52.8	60	0	0	0
60	144	1000	1000	0
144	144.24	0	0	0
144.24	144.48	495.1060289	500	0.978794
144.48	144.72	804.8939711	800	-0.61175
144.72	240	1000	1000	0
240	264	0	0	0
264	266.4	1001.714	1000	-0.1714
266.4	266.64	0	0	0
266.64	266.88	782.858	800	2.14275
266.88	267.12	0	0	0
267.12	456	1000	1000	0
456	479.95	1000	1000	0
479.95	480	0	0	0
480	482.35	0	0	0
482.35	696	1000	1000	0
696	719.95	1000	1000	0

Tables 3.6 Recovered rate and real rate in Case Study 2

Therefore, the following part has two approaches to recover the rate. One way is to correct the rest of rates with the constraint of known rate and amplitude. But this approach will break the principle of material balance. The advantage of this approach is that the assumption of linear system of reservoir is followed.

The second way is to deduce the known rate period from the accumulated production rate. The remaining periods can be recovered using the same approach as the basic case. This approach promises to uphold the principle of material balance. But it may break the assumption of a linear system.

But in this case, we choose to obey the material balance principle. Tables 3.6 show the recovered rate and error. The error is about 1 percent and most periods have the zero percent error. The errors only appear in some short periods. So, the daily rate can improve the precision of the recovered rate.

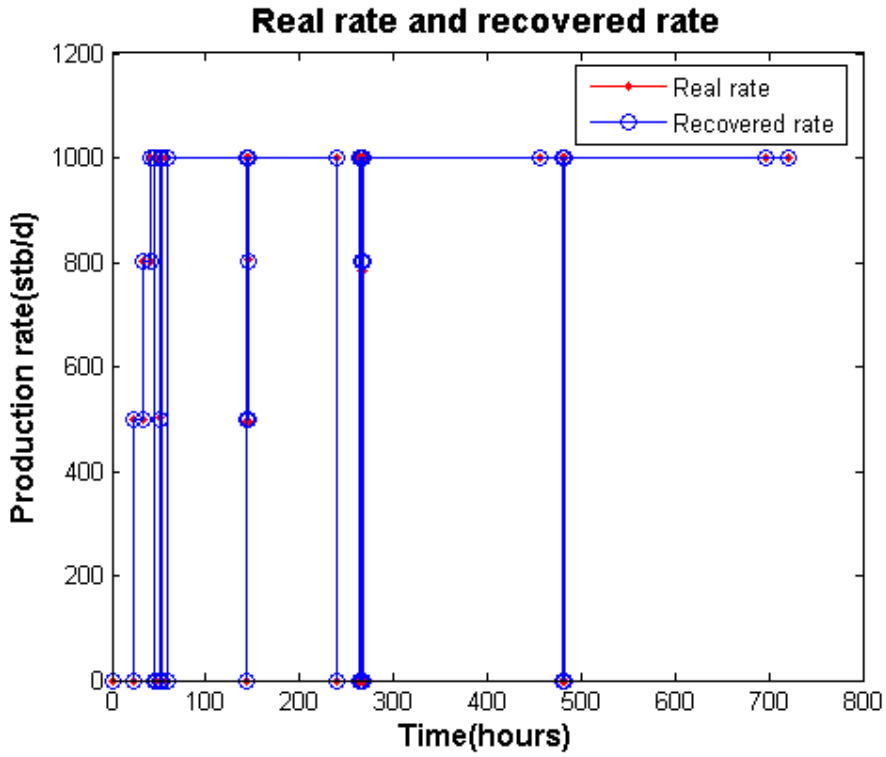


Figure 3.10 Real rate and recovered rate from accumulated daily rate

III. Case Study 3 ----Recovering rate from PDG pressure and daily production for group well

This case show a group well produced for a period of about 600 hours. The reservoir is single phase oil reservoir. The reservoir model is sandstone hestrogeneous reservoir. The two well have different skin factor and with wellbore storage. Each well is equipped with a permanent downhole gauge. Pressure measurements are acquired every 10secends. The total Rate is measured every day. Figure 3.11 illustrates the PDG pressure of the two wells.

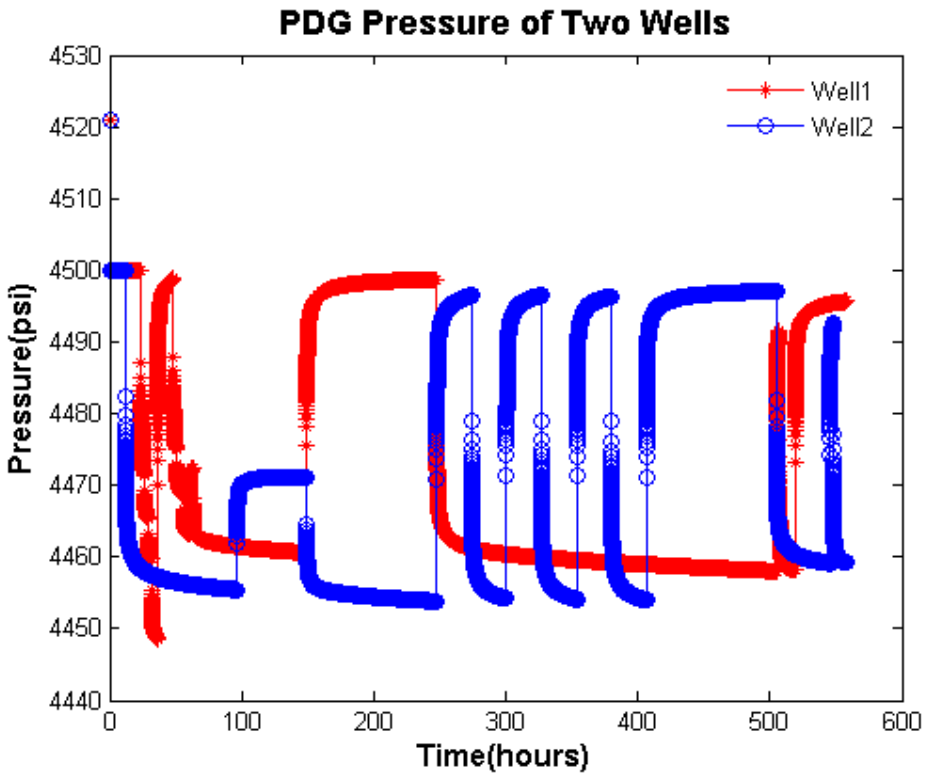


Figure 3.11 PDG pressure of two wells

According to the theory of Case Study 1, the key step to recover the production history is to find the linear relation of well. Every well have its own linear coefficient. Suppose the coefficient of m well is A_m. Then at least m equations are required to solve these m coefficients. The whole period can be divided into n periods, where n is bigger than m. In each period, the production restriction can give an equation. Suppose the accumulated time*amplitude of m well is X_{mn} in n period. And, B_n is the accumulated production in n period. Under the restriction of total production, we can write the following equation:

$$\begin{cases} A_1 * X_{11} + A_2 * X_{21} + \ldots + A_m * X_{m1} = B_1 \\ A_1 * X_{12} + A_2 * X_{22} + \ldots + A_m * X_{m2} = B_2 \\ \cdots \\ A_1 * X_{1n} + A_2 * X_{2n} + \ldots + A_m * X_{mn} = B_n \end{cases} \tag{3-17}$$

The coefficient A of each well can be solved from the least squares methods because the pressure or rate have the noise. The rate history can be recovered from the amplitude of pressure with the quation 3-16.

The recovered rates of well 1 are showed in Table 3.7. The error of every period is no

more than 2 percent. The shut-in flow period also has been properly detected. The error of first five flow period is bigger than the remained flow period because the rate variation of the first five flow period is shorter than the remained floe period. However, this result is even better than the result of Case Study 1.

Beginning (hours)	End (hours)	Real Rate (bbl/day)	Rate Recovery (bbl/day)	Error (%)
0	24.109	0	0	0
24.109	26.531	600	593.7005034	1.049916
26.531	28.953	700	691.8279155	1.167441
28.953	31.375	800	791.0027543	1.124656
31.375	36.221	1000	987.7105199	1.228948
36.221	48.331	0	0	0
48.331	50.753	500	500.4872468	0.097449
50.753	55.597	600	599.3442546	0.109291
55.597	60.441	700	698.7378435	0.180308
60.441	62.863	500	502.4289751	0.485795
62.863	149.4	700	699.934535	0.009352
149.4	247.82	0	0	0
247.82	504.77	700	701.3898662	0.198552
504.77	507.19	0	0	0
507.19	519.06	700	692.0736876	1.13233
519.06	557.35	0	0	0

Table 3.7 the recovered rate for well 1 in case study 3

Table 3.8 is the rate and recovered rate of Well 2 in case study 3. The error between two rate histories is also less than 2 percent.

Beginning (hours)	End (hours)	Real Rate (bbl/day)	Rate Recovery (bbl/day)	Error (%)
0	11.999	0	0	0
11.999	96.551	800	797.7728455	0.278394
96.551	149.4	500	505.6222038	1.124441
149.4	247.82	800	801.1007389	0.137592
247.82	274.24	0	0	0
274.24	300.66	800	797.1456656	0.356792
300.66	327.08	0	0	0
327.08	353.51	800	797.196864	0.350392
353.51	379.93	0	0	0
379.93	406.35	800	797.1712648	0.353592
406.35	504.77	0	0	0
504.77	545.48	700	697.5648554	0.347878
545.48	547.91	0	0	0
547.91	557.35	700	697.1680681	0.404562

Table 3.8 The recovered rate for well 2 in case study 3

3.3 Traditional Well testing analysis for PDG pressure

3.3.1 The procedure of traditional well test analysis for PDG pressure

There are several papers investigating how to integrate the technique of automatic well test analysis with the PDG data. The use of continuous measured data for use in well testing previous has been performed using a window approach by Athichanagorn et al. and by Khong. The main problem with this approach is that the parameters will be highly dependent upon the window size and maybe most importantly, the parameters in general show large oscillations in estimated values. Olsen presents an automatic well test analysis performed through a computer aided approach using non-linear regression with confidence intervals. All equation are given in Laplace space using the Levenberg-Marquard optimization routine with analytical derivatives. Confidence intervals are calculated for each of the unknowns.

A number of studies have been published about the topic of automatic well test analysis. Horner summarizes the four steps in computer-aided interpretation: the pressure transient data first go through a deconvolution calculation to reduce them to a would-have-been constant-flow-rate/step-change response. Then the data are presented in a pressure-derivative plot to achieve model recognition, after which the original pressure response is matched to the appropriate model using nonlinear regression. Finally, the confidence intervals are estimated to evaluate the validity of the results. The most difficult step of automatic well testing is to recognize the reservoir model for well test analysis. Various researchers have applied different approaches to these problems and a standard procedure has not yet appeared. Allian and Horne used a symbolic representation approach, achieving feature extraction by developing a sketch of the data. The symbolic approach was also used by Stewart and Du, who used a more detailed description of the extracted features. Al-Kaabi and Lee present neural networks to identify the model. Guyanguler demonstrated an approach based on a Genetic Algorithm (GA) that is able to select the most probable reservoir model among a set of candidate models, consistent with a given set of pressure transient data.

However, a true multitalented artificial intelligence program for well test interpretation is probably years away because there are more many other types of information than just pressure data, suca as geological description, drilling records and logging information. It is impossible to be able to circumvent the human interpreter entirely.

So, It is proposed here that the first analysis model setup for the basic rules for the following analysis should be determine by an engineer because it is unwise not to use the engineer's knowledge of well models, reservoir information and production history. After recognizing the reservoir model, the nonlinear regression matches pressure-transient data to the mathematical model to calculate the reservoir parameter. The objective is to minimize the sum of the squares of differences between the pressure data and the simulated data from reservoir model. The reservoir parameters are updated to achieve the smallest of the objective function.

The following parts summarize the proposal procedure of automatic well testing:

Step1: PDG data processing:

In this study, the PDG pressure is divided into different flow periods and the shut-in flow period is selected to perform well testing analysis. The further processing part would perform denoising and data reduction for PDG data. If there are production rates available for the PDG data, the following step is to directly analyze the PDG data with traditional well testing approach.

Step2: Building the analysis template:

The longest BU will be interpreted by the engineer to evaluate the reservoir model information which is then applied as a template model for the following BU. This procedure of traditional well testing includes:

Diagnose with log-log plot. Log-log plots are used to identify the different flow regimes and reservoir models required to interpret the dynamic information. The engineer first needs to decide the flow model, well model and reservoir model from the information of field. The flow regime definition in early, middle and later time regions is decided in order to classify the flow of dynamic information. Normally, the earlier regions represent the wellbore information which helps to decide the wellbore storage. The middle time regions describe the reservoir formation information, e.g. permeability. The later time regions present the boundary information, e.g. seal boundary, fault and constant pressure boundaries. The engineer needs to link the curve information with the real reservoir phenomena. Sometimes, the first model may not be not the right model, so the engineer needs to adjust the reservoir model according to the pressure in the log-log plot. Specialized analysis- performing analysis of the identified flow regimes using a specialized plot specific to a flow regime. A log-log plot gives an overall big picture of reservoir response. Specialized analysis is used to acquire the reservoir and well parameters from the specialized plot. There are just six types of

flow which can be identified in traditional well testing. These are radial flow, linear flow, bilinear flow, spherical flow, semi-steady-state flow and steady-state flow. These flow states are all straight line which can induce the reservoir parameter in the specialized plot. Simulation and matching, which include flow regime matching performed on the diagnostic plot and test history matching performed on full test history Cartesian plot. This step is applied to confirm the result from analysis of last two steps. And the automatic matching can help the engineer to obtain the optimal result.

Step3: Dynamic analysis of well testing:

As a result of the analysis of the previous step, a reservoir model as 'seen' by the test is derived with given information and engineer experience. Computers will record the reservoir model and the flow region which the engineer confirmed for the longest BU. The following BU will be regressed with the template model to produce the reservoir parameter. After the BU is completely processed, the changed parameters of reservoir are plotted to demonstrate the dynamic reservoir information. This procedure is available with current commercial well test software, to integrate the wavelet approach to process the PDG pressure data.

3.3.2 Case study of traditional well testing for PDG pressure

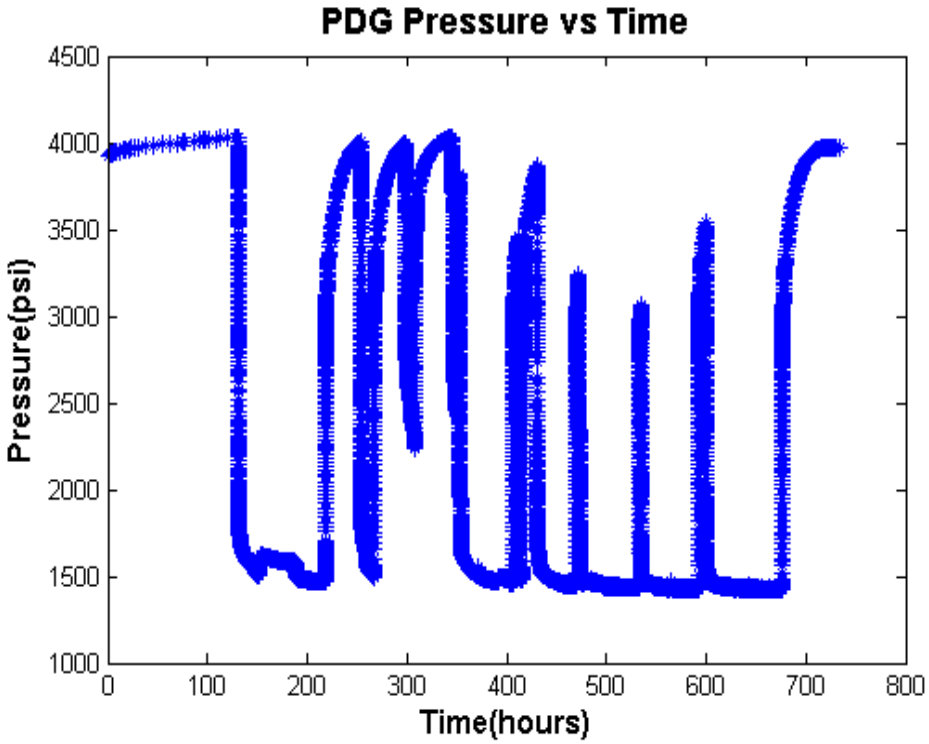


Figure3.12 North Sea one month PDG pressure data for tranditional well testing

This case is north sea gas reservoir. This reservoir is a sandstone reservoir and drive by pressure depletion. One month PDG pressure data is choosed to demonstrate the approach to analyze PDG pressure with tranditional well testing. PDG is installed to record the pressure and temperature every 5 seconds. The production rate is recovered from PDG pressure and total production rate. After rate recovery and data processing, this one month PDG pressure data is plotted in figure 3.12. This data has been processed by the wavelet transform method to distinguish the whole PDG data into BU and DD.

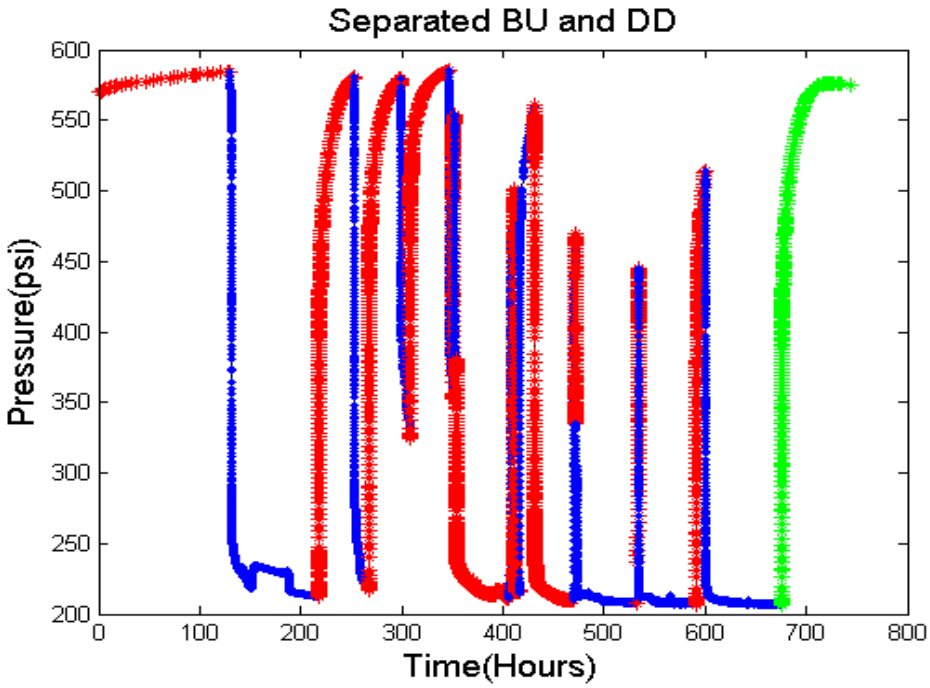


Figure3.13 Separated BU and DD for tranditional well testing

Figure 3.13 show the separated BU and DD after the PDG data processing of wavelet approach. There are ten shut-in BUs. These BUs will be used to perform well test analysis. The longest BU will be used to perform analysis in order to obtain the template reservoir model.

Figure 3.14 is the loglog plot of the longest BU. The loglog plot is used to diagnose the reservoir model and different flow region. We use the radial homogenous and seal boundaries to match the pressure data according our experience. In this short term data, there are only two flow regions. The second step to perform analysis is to calculate the skin factor and permeability from semi-log plot which is calculated from figure 3.15. This model will be stored as the template model for the remaining reservoir models.

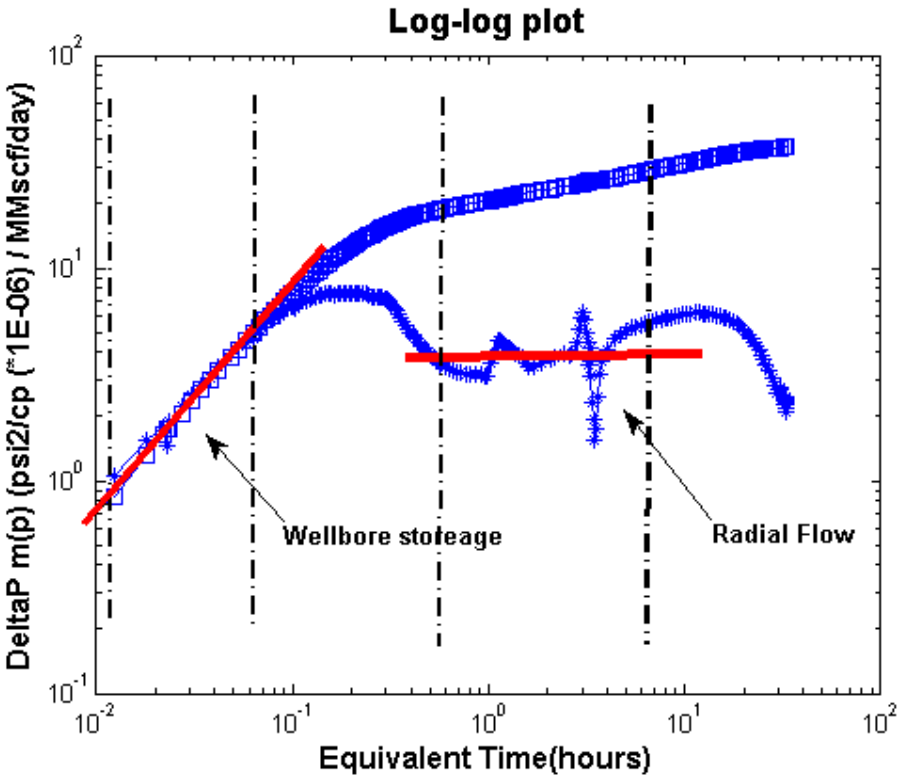


Figure 3.14 Log-log plot of BU for tranditional well testing

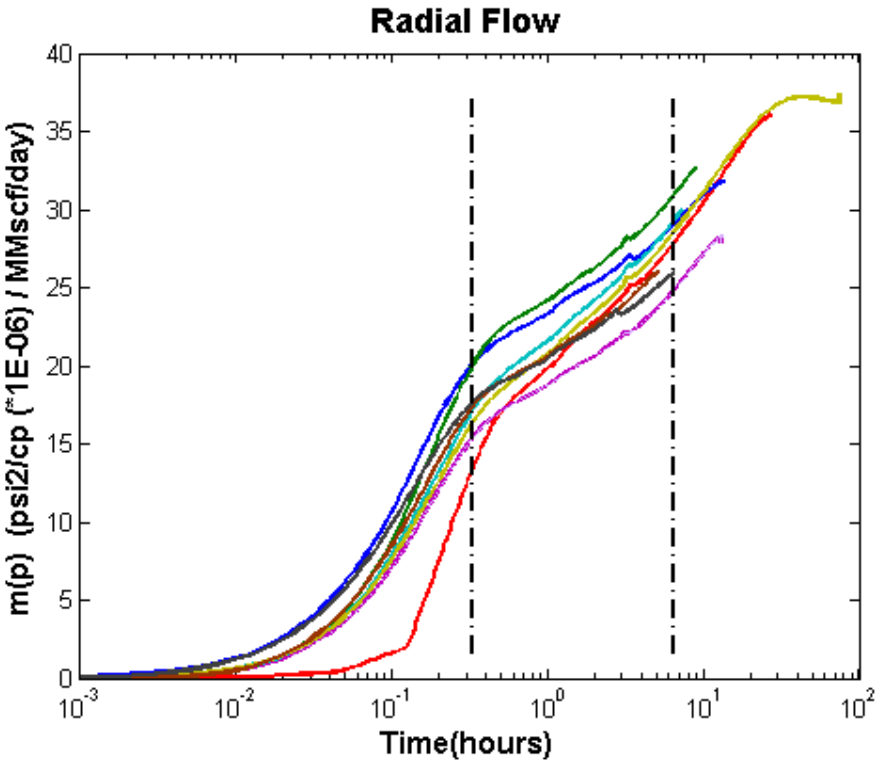


Figure 3.15 Semi-log plot of all BU for tranditional well testing

Figure 3.16 shows the changed reservoir parameters calculated with nonlinear regression with the template model for every BU.

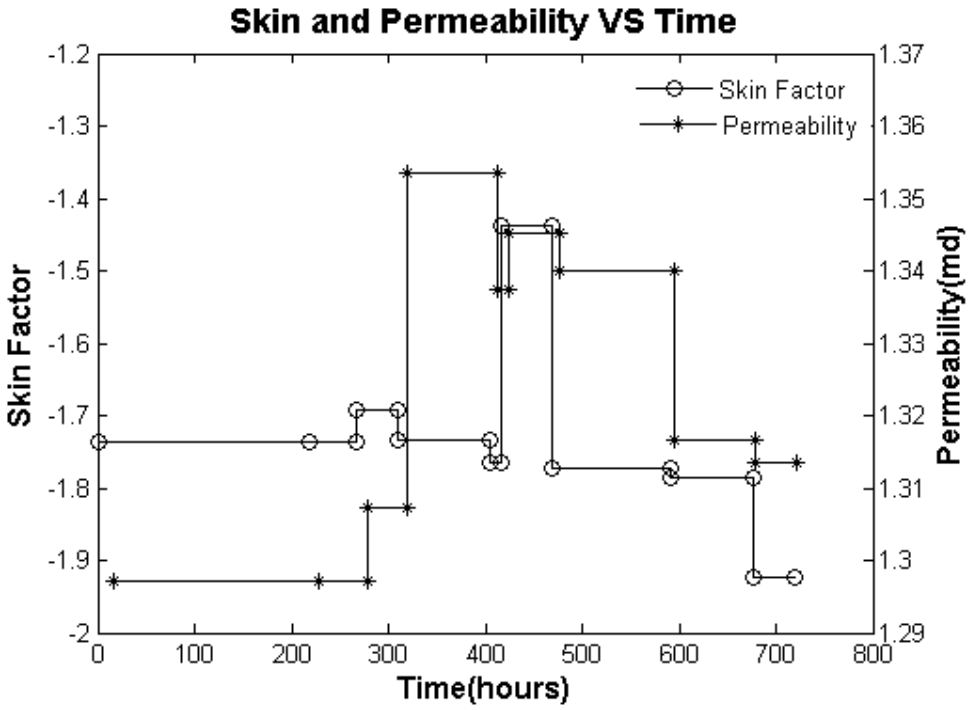


Figure3.16 Changed reservoir parameters from tranditional well testing

3.4 Numerical Well Testing analysis for PDG pressure

The NWT technique has been developed to tackle well testing problems in heterogeneous reservoirs. The traditional analytical approach can only solve the ideal reservoir model. The numerical approach can solve the more complex reservoir model. But the uncertainty also increases with the numerical model. This section will discuss how to apply numerical well-testing for PDG pressure.

Several researchers have begun to apply the NWT in different cases. Jackson presented an approach to analysing multilayer reservoir testing using numerical well testing. Pinzon (1987 [70]) studied the influence of a stress sensitive reservoir with NWT. Warren (1993 [89]) presented a numerical solution involving partial penetration and non-Darcy flow turbulence. He first briefly reviewed the development of the girding and time interval control algorithms and discussed the advantage offered by numerical models which he compared to standard analytic solutions. Zheng (1996 [128]) was the first to integrate geology and well testing information for fluvial reservoir characterization. He made a systematic approach which progressed logically from the disciplines/methods of fluvial geology fluvial reservoir sediment logy, formation petrol physics, modern/ancient fluvial channel assessments, numerical/analytical modeling/simulation to well test analysis/interpretations. In detail,

this method includes three steps: definition of an interpretation model – this requires the integration of geological, seismic and petrol physical data with transient pressure data; analysis of the well test data based on the interpretation model defined; and geological interpretation of the results which is necessary in order to explain or give meaning to the results. Vicente (2005 [112]) developed a fully implicit, 3D simulator with local refinement around the wellbore to solve reservoir and horizontal well flow equations simultaneously for single-phase liquid and gas cases. Kamal (2006 [68]) proposed a step-by-step approach of incorporating well test data into a full-field simulation model using transient analysis standard. This section first introduces a moval procedure of using the NWT for PDG pressure data. A case study then demonstrates the procedure.

3.4.1 Procedure for applying NWT for PDG data

The procedure of applying NWT for PDG data includes two parts. The first part is to prepare the geology model, fluid model and dynamic production and pressure information. The second part conducts a numerical experiment for well testing analysis in order to update the reservoir geology and fluid model. The following procedure is based on the application of ECLIPSE.

Preparation for NTW first generates a near well model with local grid refinement from the full field model. In addition, the production information of well is reallocated with refined time interval for NWT.

Acquiring the proper reservoir geological model is the crucial step for NWT. Normally, the full field reservoir geological model is supplied by the geologist who integrates the seismic information, geology information and log information with geostatistics model. However, the full field is too large for well testing analysis and it also consumes a large amount of computer time. It will therefore be more efficient to extract part of full field reservoir model to analyze the well transient information. The Near Wellbore Modeling (NWM) module in ECLIPSE allows the user to generate a detailed local model around one or more wells in an existing full-field ECLIPSE model. The flow between the full field model and part model can be also considered. The area of influence of the well is extracted from the analytical model according to the flow boundary.

The downhole pressure calculated from simulation software is equal to the average pressure of the global grid. And the global grid size is normally from 30 feet to 50 feet. In addition, the true downhole pressure is the pressure near the wellbore which is much

smaller than the global grid size. Hence the simulated downhole pressure cannot represent the true bottomhole pressure of reservoir. The grid model should be refined to match the real performance of well. Accuracy and efficiency of BHP in complex systems are highly dependent upon a proper grid selection. Grids based on a Cartesian coordinate system have been widely used, but the Cartesian grid has no flexibility in description of faults, pitchouts, hydraulic fractures and well location. Researchers have introduced local grid refinement to enhance the performance of Cartesian grids. The local models can be 2D radial, 3D radial or 3D Cartesian. Transmissibility between the local models and the global model are computed automatically by software. The properties of the cells in the local grid can be inherited from the Global grid or specified explicitly for the refined cells. In this study, the mixed grid is refined. The mixed grid selects radial local grid refinement for the grid where the well located and the grid around the well is refined with Cartesian grid.

After the proper grid model is built, the model is run to simulate the transient pressure. Preparing pressure and production data is also an important step for NWT. The NWT requires solving an inverse problem. This involves matching huge amount of observation data while changing numbers of reservoir parameters. Normally, the production history will be fed into the reservoir model to simulate the pressure data. The reservoir model will be validated by comparison between the simulated pressure data and observed data. It is necessary to confirm that the input dynamic is right or at least reasonable. So there are two issues to be discussed to confirm the right input for the model. The first is to input the right production history for matching pressure. Normally history matching always allocated the production rate into each item of data for simulation of the schedule file. However, there is no way to match the real time PDG pressure with this type of production history. The accurate data is required during numerical welltesting.

The second issue is to verify the PDG data. The PDG data is recorded as real time data under natural environmental conditions. These data definitely reflect the real performance of reservoir, the influence of wellbore, operation and measurement errors of gauges, but the ideal numerical simulation model cannot reflect the total real information of the reservoir. Therefore it is important to preprocess the PDG pressure data before applying it to perform history matching. As regards system and signal, the high frequency information are connected with well information, noise and operation events. The low frequency signals are connected with the reservoir information. The

simulation model can only demonstrate the reservoir performance in the normal workflow.

The SCHEDULE section specifies the operations to be simulated (production and injection controls and constraints) and the times at which output reports are required.

The schedule files of normal history matching are based on daily production rate and constant well control. These schedule files cannot reflect the real response of the reservoir. The alternative choice is to allocate a time interval of the production rate with an equal to the PDG pressure time interval. Hence the production data and real pressure data have a one-to-one relationship. The drawback of this method is that the schedule file will include large amounts of time-steps which results in a huge simulation time. A more effective approach is to reduce the data points of original pressure data without losing the reservoir information. The data points in the low frequency range will be sampled in a large time interval. The data points in the high frequency range will be sampled in a small time step. We want to compare the pressure data in different plots (Cartesian plot, simelog plot and log-log plot). The production time step of every flow period will be allocated equal space in the log scale.

A NWT is proposed to preserve the information obtainable from traditional well test analysis and deliver it in a form suitable for direct use in numerical reservoir simulation. By using the numerical method, engineers can use transient testing results to incorporate averaged values in their discretized reservoir model. NWT is an inverse problem. It is the same as the history matching of reservoir simulation. The difference between the two problems lies in scale and object. NWT uses the near well region to perform simulation. NWT matches the pressure in different plots. The reservoir simulation applies the full field model to perform matches, and simply matches the pressure in the Cartesian plot. The core of NWT is to obtain the right model to reflect the real dynamic information produced by the reservoir.

The bottleneck of numerical welltesting is forward modeling. The difficulty of forward modeling is to fight against non-unique problems. The whole idea of forward modeling is to try to modify the parameters of the model to match the real response of a real reservoir. The first problem of forward modeling is how to change the parameters of the model. There are thousands of parameters which can be changed. It is necessary to know the relationship between the parameter variations of the model and the response of the system. The second problem is what type of criteria are used to confirm the match. As we know, the response of the real world can never be simulated

in the computer. But it is also helpful to give a form of approximation. In order to perform history matching, thousands of experiments may be produced to match the real data. It is impossible to handle these manually. So, it is necessary to develop a toolbox to integrate current commercial software to perform the NWT. This study develops a toolbox with Matlab and ECLIPSE. The toolbox includes three components: the first part is used to design the numerical experiment and send it to the simulator for calculation. The second part is to analyze the results from numerical experiments to obtain the general principle for the updated model. The third part is to update the original model according the principles and conditions set by the operator.

3.4.2 Designing the numerical numerical experiment modular

The normal workflow when designing a numerical experiment is building the model, changing the sensitivity of the parameters and running the simulation. This cycle is easy and quick for a few numerical experiments. Suppose we have two hundred models that need to be run. It is impossible for one person to achieve this task, and it will definitely be more than this to obtain the right model in forward modeling. Therefore this task should be resolved on a computer.

One of the most essential links in this cycle is the simulator. But it is impossible for us to develop a new simulator to perform calculations. Instead it is suggested benefit from the commercial software applications which are already available. ECLIPSE is selected for this purpose. This section will describe how to link ECLIPSE to numerical experiments for forward modeling.

The procedure of an automatic numerical experiment follows five steps. The first step is to build ECLIPSE data for a certain problem. The second step is to read the ECLIPSE data file and detect the parameter which will be changed for sensitive study. The third step is to load the experimental plan from the user interface. The fourth step is to write a new ECLIPSE data file with the changed parameters. The fifth step is writing the batch file to run the ECLIPSE data file for calculation.

A small case study is used to demonstrate how to perform numerical experiments with our tools. Suppose we already have an ECLIPSE file. This is a radial flow model. And we simply want to change one-ring permeability to perform a sensitive study. The ECLIPSE data file is written like this:

--&&& change one ring

@BOX

$40 $40 1 $6 1 1 /

-- Permeability

PERMR

$6*$5

/

#ENDBOX

@ marks the Beginning of keywords which are modified. Here this means that some part of keywords **BOX** will be changed. # is the end mark for the changed keywords. $ means the following value is designated to be changed for experiment. In this case, the changed keywords are BOX. There are five parameters which can be updated for new models. The first two parameters involve the ring number; the third and fourth numbers give the number of the section of the ringand the last one is the permeability in the radial direction.

The software reads the marked data file. The changed keywords and parameter will be detected and displayed in the interface. The user can assign the group member from the parameter identified from the original eclipse file. The range for every group is give by the user. For example:

Group1:

Member = [1 2]; Range = [20 20; 40 40; 60 60; 80 80; 100 100];

Group2:

Member = [3 4]; Range = [1 1; 2 2; 3 3; 4 4; 5 5; 6 6];

Group3:

Member = [5]; Range = [5; 20; 50; 100; 500];

Group Array:

[Group1 Group2 Group3];

There are three groups in the experiment. Group 1 contains two members. These two parameters will be changed at the same time. This means the permeability changes from ring 20 to ring 100. Group 2 contains two members. These two parameters will also be changed at the same time. This means the permeability will vary from one section to six sections of the ring. Group 3 contains one member. And this parameter changes the permeability five times. So the number of this experiment is 5*6*5 = 150. Then the 150 eclipse data file will be written and run for simulation.

The second component of these tools analyzes these hundreds of results and obtains the principle of the reservoir. The dynamic data is plotted in different plots in order to

check the response of the reservoir to the changed parameters in different views. The user can select the analysis type and the result of the experiment. There are three types plot – Cartesian plot, Semi-log plot and log-log plot.

The procedures of automatic numerical experiment analysis comprise three stages. The first stage is to obtain the requirement from user interface. The user can select the analysis type and the result of experiment to compare the effect of different varieties. The second stage is to load the experiment from the ECLIPSE *. RSM file according to the conditions set by the user. The format of ECLIPSE *.RSM is regular. There are no more than 10 data columns in one line. There are three lines to allow comment on the data before the data. The first line consists of the keywords from the ECLIPSE summary part. The second line is the unit for every keyword. The third line is the assigned name for keywords. The third stage is dynamic pressure analysis. Three plots can be displayed at this point. Multi-data can be plotted in the same plot to compare the difference between the changed parameters and summarize the principle of the reservoir response.

This stage continues the experiment of the last small case study. Suppose you want to check the permeability that changed from 5 to 5000 in the 20 ring. You need to input the follow parameter into software. For instance:

Group1:

Member = [1 2]; Range = [20 20];

Group2:

Member = [3 4]; Range = [6 6];

Group3:

Member = [5]; Range = [5; 20; 50; 100; 500];

Group Array:

[Group1 Group2 Group3];

The plot will display the pressure of different permeability case in a single figure. The sensitivity parameter is loaded from the result of the last step. The sensitivity coefficient of the elected parameters is calculated using the Gradient method. Then applying automatic history matching techniques in a regression program, these properties are modified until an acceptable match is obtained between the numerically simulated results from the model and measured pressure and pressure in different plots. Normally, there are three types of plot: Cartesian plot, special-analysis plot and log-log plot.

The criterion for confirming the matched parameter is not just decided from one plot. It follows three levels. At the basic level the Cartesian plot must be matched. Then the specialized analysis plot must be matched. The matching of the log-log plot is the most difficult part. Therefore, the log-log plot matching must be confirmed after the matching of the other two plots has been finished.

3.4.3 Case studies of numerical well testing for PDG pressure

These case studies concern the PDG pressure from a North-sea gas condensate reservoir. The target field is a large gas condensate reservoir which occupies about 250 km^2 (98 $miles^2$). The reservoir porosity of reservoir is about 15%. The permeability ranges from 0.1 to 50 mD. And the vertical depth of well is 13000ft. The gross reservoir thickness is about 300ft, Net To Gross 12-60%. The initial pressure of the reservoir is 5990psia and the dew point is 5600psia. The drive mechanism is pressure depletion.
The reservoir is a deep water turbidity reservoir. The well is located geographically between two areas where missing or thinned sands are correlated within the time-equivalent shale sections. The poor development of the sands is most likely related to sub-positive basin topography. Reservoir variability is also caused by offset stacking of individual sand bodies where the slight topographic bottom features of early sand accumulations within the turbidity turbidite fairway influence and deflect successive flows resulting in sand thickness changes both laterally and vertically. The reservoir model is built on the ECLIPSE 100. The size of the full field is 180*74*56. It is approximately about 745920 cells. The full field model is too big for performing the numerical simulation of the well. So, the area around the well 15 is selected, according to the formation properties and the distance between these wells and well 15. The area sensitivity analysis is then applied to obtain the area model. The flow state of these wells, especially well 15, will be the key factor to select the area model. The figure 3.17 shows the chopped area.

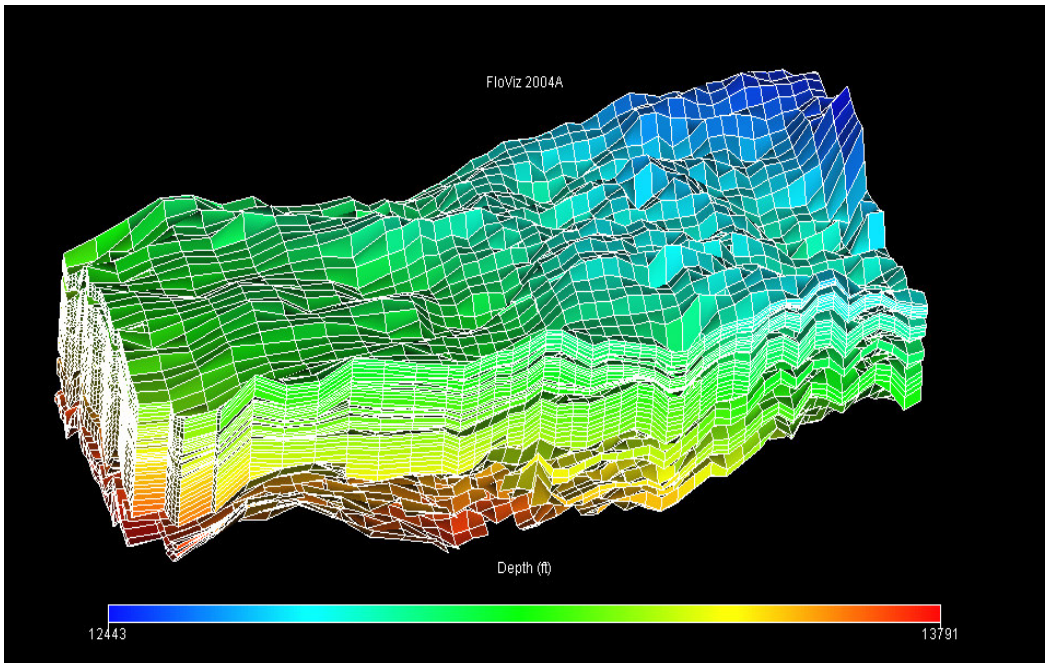


Figure 3.17 Area model around well for numerical well testing

In order to simulate the flow state near the well, the Local Grid Refinement of well 15 is applied. The Local Grid Refinement option allows enhanced grid definition near wells. The local models may be 2D radial, 3D radial or 3D Cartesian. Local models may have more layers than the global model. This study applies the radial local grid refinement at the grid where the well is located. The Cartesian grid is applied around the well. Figure 3.18 show the radial LGR.

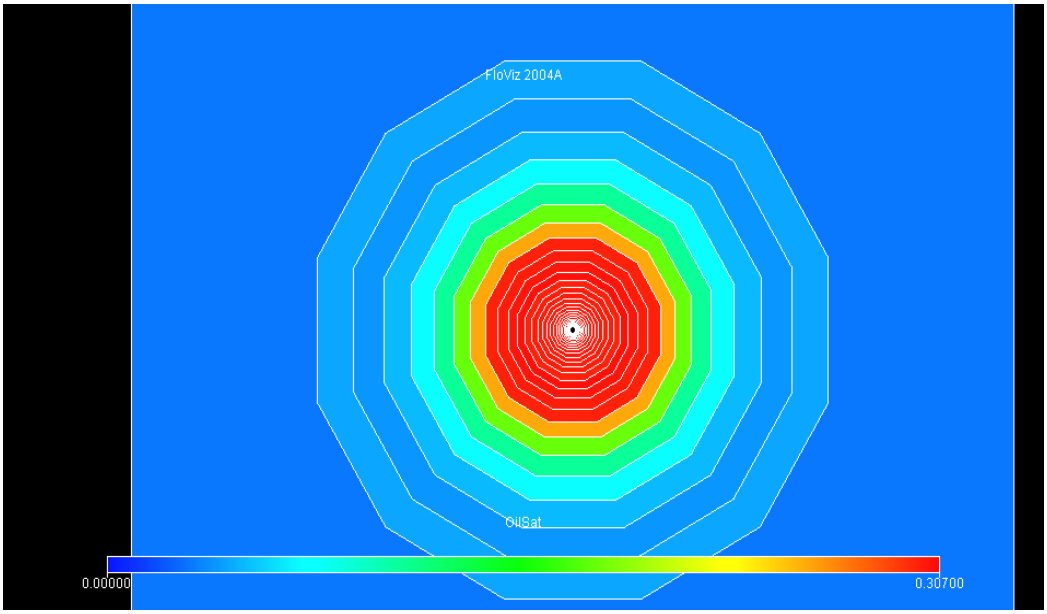


Figure 3.18 Radial LGR for numerical well testing

2 x Omega short quartz gauges are installed in well B. The 'compig' system is

installed to monitor the data. The Gauge depth is 13158 feet (MD). The PDG was inserted into the well and the data is transferred by wire-line to the surface. The record frequency is set as five seconds. There is no memory at the wellhead, so these data will be sent to the computer and saved in the database of company. However, the frequency of collected PDG data in the database is not 5seconds because the database software ignores some changed data in order to save memory space. In order to perform numerical well testing, the first 21 days data is selected to analyse. The real time downhole rate is from the downhole flow meters. However, the quality of this data can not meet the requirement of numerical well testing. The normalized step and data processing step is operated to obtain cleaner and clearer data. Figure 3.19 shows the PDG pressure data and the rate history.

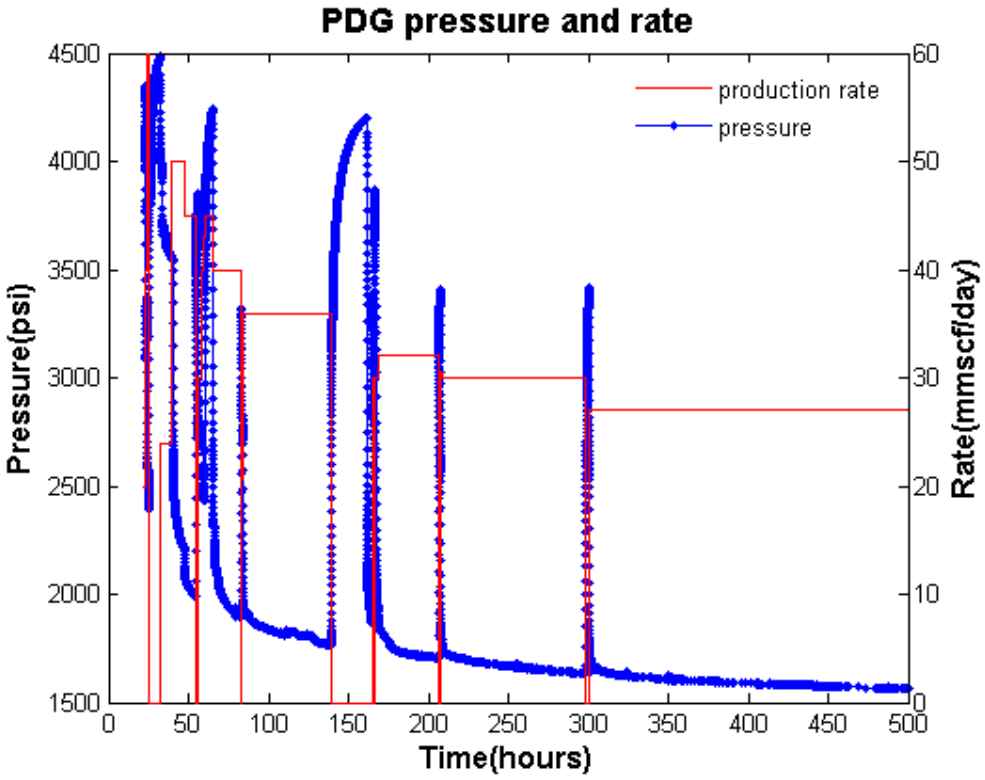


Figure 3.19 PDG pressure and rate of well 15 for numerical well testing

After preparing the dynamic data and static reservoir model, the whole dataset can be run with ECLIPSE. The simulated downhole pressure is produced from ECLIPSE. The numerical well testing utilizes the simulated data to match the real pressure data in different plots. This study first tries to match the PDG pressure data in a Cartesian plot in order to obtain the first estimates about the reservoir. Then the focus will be on the longest BU so as to acquire the reservoir properties. Figure 3.20 show the loglog plot

from the longest BU of real PDG data. This plot can be divided three parts. The early part is the wellbore storage which may be affected by the phase redistribution. The middle part is radial flow which is also affected by phase changing. The later part of this plot can not reveal a reasonable signal for the boundary because the test period is not long enough. It may be the boundary but it could also be an infinite boundary. So, the traditional well testing can only match the middle part of the model with straight line to obtain the radial flow. However, NWT can give more options to match the real data.

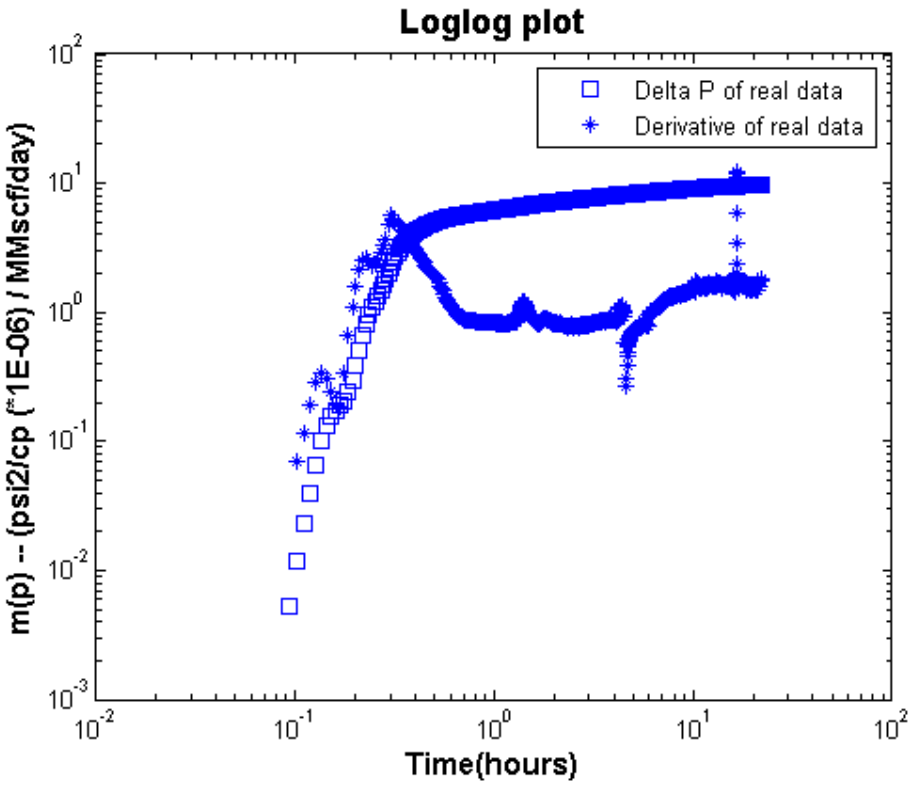


Figure 3.20 Log-log plot of longest BU with real PDG data

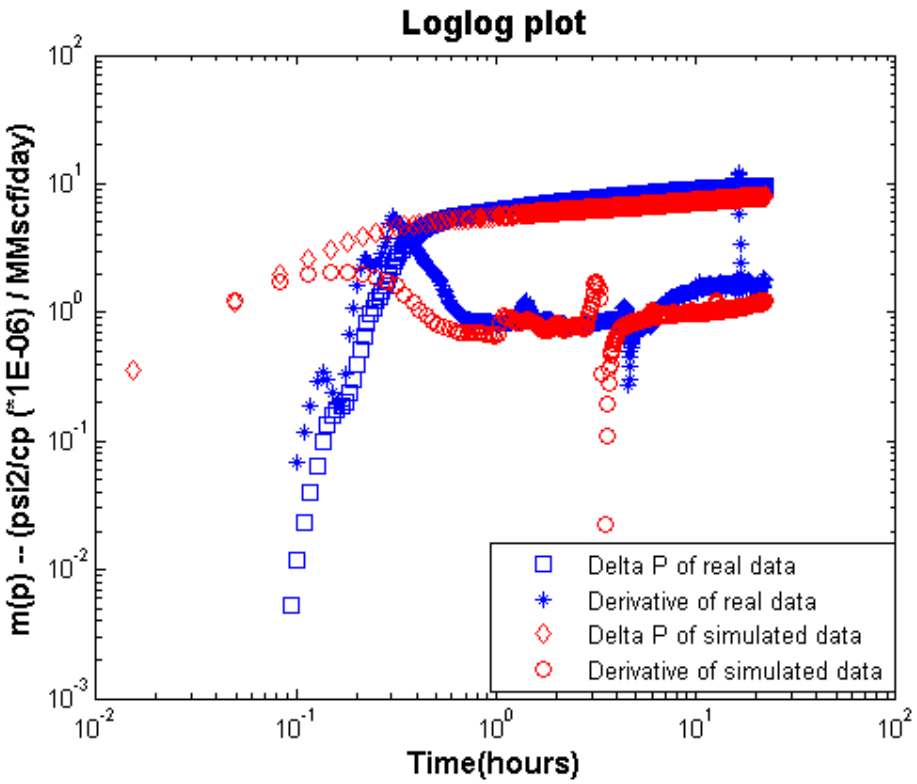


Figure 3.21 History matching of log-log plot

With the help of the numerical experiment tools, figure 3. 21 is one of the best matching from simulation. In this graph, the early region did not match the real data because the reservoir simulator can only demonstrate the classical wellbore storage. However, the middle part of data has been matched. Not only did the radial flow match, but also the fluctuation among the radial flow, caused by the reservoir phase changing when the reservoir pressure changes, is matched. And the later part also shows the same trend. So, this picture can be confirmed to match the real dynamic information. And the chopped model is also restored to the full-field model.

3.5 Chapter conclusion

The first part of this chapter introduce a new approach to carry out producton rate recovery from PDG pressure together with accumulated production. This new approach assume the reservoir system is a linear system in short time period in order to get the linear relationship between rate variation and pressure vaiation. This noval approach has been tested with three casees. We found this approach can work very well in single phase oil reservoir. This approach also can be extend to group well case and daily rate case. However, this approaches will produce big errors when the flow

event did not produce for long time. This approach also increase some errors when the reservoir is nonlinear system or the well have big wellbore storage effect.

The subsquence part of chapter present how to analyze the PDG pressure with traditional well testing approach. In traditional well testing approach, a PDG pressure of gas reservoir is analyzed to get the changed parameters. We found the data processing approach in chapter also can process the gas reservoir and gas condensate reservoir pressure data. Although the changed parameters of reservoir can be calculated from tranditional well testing, the variation of reservoir parameter still can not be confirmed because the DD are not analyzed.

The last part of chapter discuss how to analyze PDG pressure with numerical well testing approach. A new workflow is developed for PDG pressure numerical well testing. In order to work more efficiently, a toolbox has been developed to realize the numerical well testing with the current ECLIPSE software. A gas condenstate case is studied to analyze the PDG pressure. The numerical well testing approach can match more complex reservoir geological model and reservoir fluid model. It also can consider the history information of reservoir. However, the reservoir system are still considered as a static system.

CHAPTER 4 DYNAMIC FORWARD MODELING WITH PDG PRESSURE DATA

4.1 Introduction

The previous chapter discussed how to apply the traditional well testing and numerical well testing to analyze PDG pressure data. The traditional well testing approach can obtain the changed reservoir information from the separated BU. However, this approach omits the reservoir information which comes from DD of PDG pressure data. The numerical well testing approach has the advantage to analyze the complex geological model. But, it still focuses on the part information of PDG pressure data. Recently, the deconvolution approach can transfer the multi-rate pressure into a constant rate response of the reservoir. This approach doesn't break the whole PDG pressure data into pieces. This approach will obtain more information from the whole PDG pressure data. However, the deconvolution approach can only be applied in the linear system. The real reservoir system is a dynamic nonlinear system which may be caused by multi-phase problem, multi-well problem or changed parameter of reservoir. Therefore, it's necessary to develop a new approach to analyze the nonlinear PDG pressure data.

In this chapter, a new approach is addressed to analyze the nonlinear PDG pressure data. This approach integrates the deconvolution approach, traditional well testing approach and numerical well testing approach. The first part of this chapter discusses how to diagnose a dynamic system with deconvolution algorithm. The second part of this chapter is a case study to demonstrate a three steps procedure to analyze the nonlinear PDG pressure data. The traditional well testing result is treated as the first guess for linear system detection. The deconvolution method is applied to separate a dynamic system into a piecewise linear system. Each piecewise static model is analyzed with the numerical well testing approach to reveal the dual character of PDG pressure data.

4.2 Diagnosing a nonlinear system using the deconvolution method

The PDG data recorded from nonlinear system can not be analyzed with current approach because the theory of the current approach is based on the linear system. There are two approaches to analyze the nonlinear system data. The first approach is to build a complex nonlinear model for the real reservoir. The second approach is to linearize the nonlinear system into separated linear systems. This study considers the second approach because the nonlinear model is hard to build and the nonlinear equation is not easy to solve. The key step to linearize the nonlinear system is to divide the nonlinear system into separated linear system. The deconvolution is applied to realize it because it reveals some discontinues in deconvolution result when one linear reservoir system change into another linear reservoir system. The subsequence part will introduce how to use the deconvolution approach to diagnose the nonlinear system.

4.2.1 Theory of deconvolution

The deconvolution is to acquire the constant system response. It can be represented as the unit response, impulse response or other kind system response. In reservoir system, deconvolution is to find the relationship between input production signal and the output pressure signal of well. The deconvolution approach has been used to transfer multi-rate PDG pressure into single rate response of a reservoir (Hutchinson, 1959[108]; Coats, 1964[55]; Jargon, 1965[52]; Kucuk, 1985[40][41]; Thompson, 1986[63]; von Schroeter,2004 [111]; Cheng, 2005[122]; Levitan, 2005[69]; Sanghui, 2008[98]).

Three directions of deconvolution algorithm are presented in this study. The first approach is to calculate the unit rate response of the reservoir. The second approach is to compute the impulse response of reservoir. The third way is directly to obtain the log-log derivative reservoir response.

In a linear system, the well pressure during a variable-rate test is given by the convolution integral.

$$p(t) = p_0 - \int_0^t \dot{Q}(\tau) p_u(t-\tau) d\tau \tag{4-1}$$

Here $\dot{Q}(t)$ is the well rate derivative, $p(t)$ is the well bottomhole pressure, and p_0 is the initial reservoir pressure. The p_u in equation 4-1 is the rate normalized pressure

response to constant-rate production, assuming that, at the beginning of production, the reservoir is in equilibrium and the pressure is uniform throughout the reservoir. Eq.4-1 is known as Duhamel's integral and is an expression of the principle of superposition resulting from the linear character of the system. The deconvolution is to calculate the constant rate response of system p_u.

Equation 4-1 can be written like equation 4-2. Here g denotes the impulse response, which is the ordinary time derivative of rate-normalized wellbore pressure drop. The pressure drop Δp is the observed pressure in well test with time varying production rate $Q(t)$.

$$\Delta p = \{Q * g\} := \int_0^t Q(\tau) g(t-\tau) d\tau \tag{4-2}$$

The deconvolution in equation 4-2 is to calculate the impulse response of the system. Recently, von Schroeter et al. (2004 [111]) presented a new equation to express this linear relationship. He use the pressure derivative to represent the constant relationship between pressure drop and varying production rate.

$$\gamma(t) = \frac{dp_u(t)}{d\ln t} = tg(t) \tag{4-3}$$

$\gamma(t)$ is the pressure derivative. The equation 4-3 can shift to equation 4-4. Duhamel's principle becomes ,

$$\Delta p = \int_0^t Q(t-\tau)\tau\, g(t-\tau)\frac{d\tau}{\tau} = \int_{-\infty}^{\ln t} Q(t-e^{\tau}) e^{Z(\tau)} d\tau \tag{4-4}$$

Here , suppose $\tau = \ln t$ and $Z(\tau) = \ln(tg(t))$. From equation 4-4, the deconvolution problem changes to solve $Z(\tau)$.

Deconvolution aims at reconstruction of the constant system response p_u, g and $Z(\tau)$, along with the initial pressure p_0, from the pressure and rate data, acquired during a variable-rate PDG data. A conventional method for solving equation has been the method of least squares (LS). The underlying assumption in the solution of the ordinary least squares problem is that errors only occur in the output parameter and that the input parameter is exactly known. Often this assumption is not realistic because of sampling, modeling or measurement errors. Current way to take errors in both sides into account is to solve the Total Least Squares (TLS) problem. After reviewing the current deconvolution algorithm, the following part will discuss how to diagnose the

nonlinear reservoir system.

4.2.2 Appling deconvolution for the nonlinear system

As we known, the deconvolution approach can only be applied for the linear system. However, the real reservoir system is always a nonlinear system. This part will discuss what occurs when current deconvolution approach is applied for the nonlinear system.

A nonlinear system is a system whose parameters change with time. The parameter may change very slowly, e.g. permeability. Some other parameters may change very fast, e. g. skin factor and wellbore storage. In this research, we mainly consider one kind of nonlinear system which the system parameters have the step variations.

A synthetic nonlinear system is simulated with Eclipse. This reservoir is a multi-layer heterogeneous oil reservoir with single well. The reservoir model has 50*50*10 grids with local grid refinement of the well. The well produces with 800bbl/day for 44 hours. The downhole pressure gauge is installed to record the pressure in every 5 seconds. This well has wellbore storage and the skin factor changed from 5 to 4 at 29hours. In addition, the record errors are also considered on the pressure gauge and flow meter. It is important to note the error model that we use for pressure and rate data. We consider independent, identically distributed normal random errors with zero mean and specified standard deviations for pressure and rate data. Figure 4.1 shows the simulated pressure and rate data in a nonlinear system with noise.

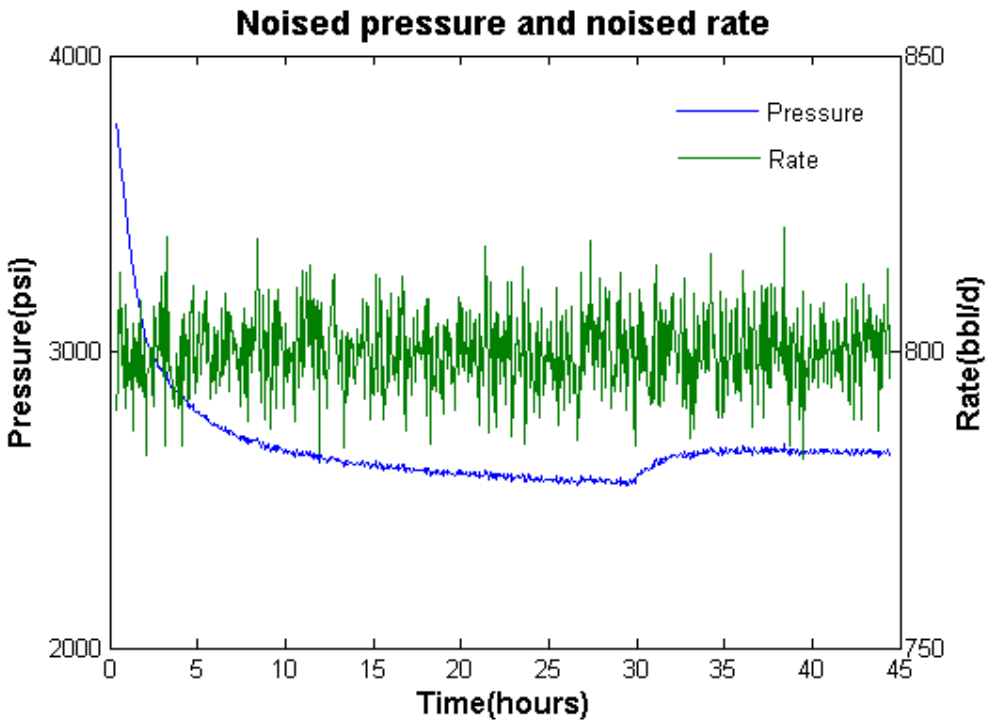


Figure 4.1 Noised pressure and rate of nonlinear systems with changed skin factor

The simulated pressure and rate data all have noise, outliers and other fluctuations. It is impossible to calculate the deconvolution result from the noise data because the direct recursive approach is numerically instable, as Hutchinson and Sikora (1992 [108]) have already observed. Coats et al. (1989 [55]) were the first to give another way to reformulate deconvolution as an optimization problem. The problem is to transfer data using methods of least squares and constrain to calculate the minimum errors. Recently, von Schroeter et al. (2004 [111]) have published an approach using the nonlinear Total Least Squares (TLS) with constraints to perform deconvolution. The total least square method considers the errors of rate and pressure. Here we follow this idea to calculate the impulse response, unit response and the log-log derivative from the noise data. In the following part the deconvolution results of a nonlinear system from these three approaches are compared.

The first approach is to calculate the unit response of system. The deconvolution equation gives the original relation between the pressure, rate and unit response. And this equation can be written into equation 4-5 with noise in rate and pressure.

$$\Delta p + \varepsilon = \int_0^t (\dot{Q}(\tau) + \delta) p_u(t-\tau) d\tau \qquad (4\text{-}5)$$

Considering the noise in rate and pressure, the optimal problem is applied to solve a TLS problem with constraints. Hutchinson and Sikora argued on physical grounds that

the unit rate response function should not only be positive and increasing, but also concave in a linear system. The normal unit rate response of a linear system should satisfy the following condition:

$$p_u \geq 0, \dot{p}_u \geq 0, \ddot{p}_u \leq 0 \tag{4-6}$$

Where, p_u is the unit rate response of system, $\dot{p}_u$ is the first order derivative, and $\ddot{p}_u$ is the second order derivative. This constraint assures the unit rate pressure is continue and goes up.

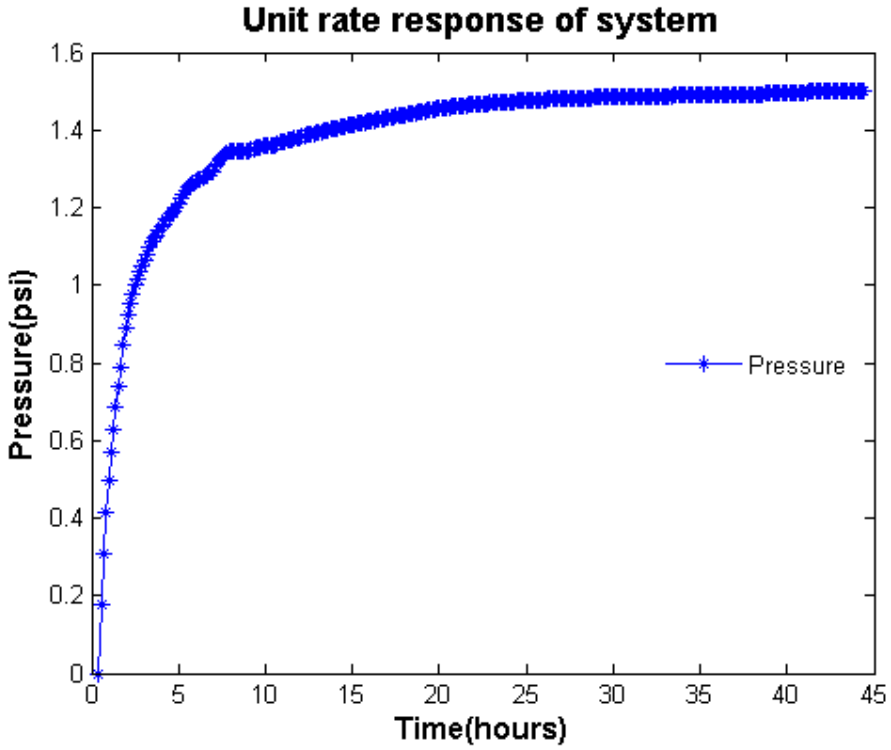


Figure 4.2 Unit rate response of noised nonlinear system after deconvolution

Figure 4.2 illustrates the unit rate response of system. We find that the curve is very smooth and the trend goes up. But there is no variation at 29 hours. The nonlinearity of system cannot be detected because the constraints and optimal algorithms remove the variation of the nonlinear system. This result will cause a wrong analysis.

The second approach is to calculate the impulse response of the system. The deconvolution equation gives the original relation between the pressure, rate and impulse response. And this equation can be written into equation 4-7 with noise in rate and pressure.

$$\Delta p + \varepsilon = \int_0^t (Q(\tau) + \delta) g(t-\tau) d\tau \tag{4-7}$$

Equation 4-7 can be solved with TLS algorithm with constrain. This constrain

confirms the impulse response is positive because there is no negative pressure value in real reservoir system.

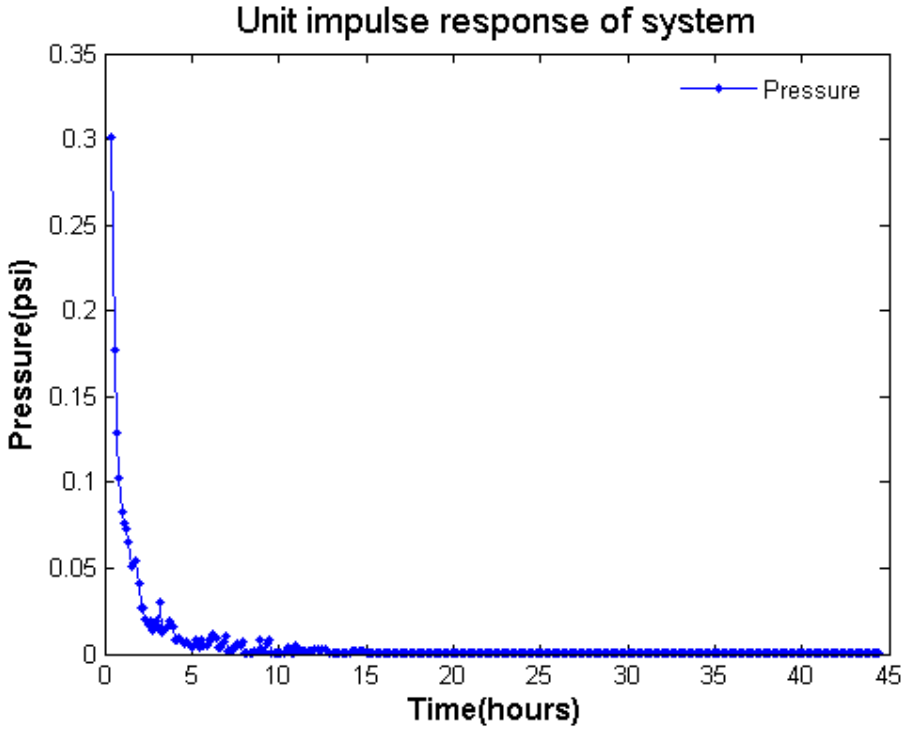


Figure 4.3 Unit impulse response of noised nonlinear system after deconvolution

The blue curve in Figure 4.3 is the unit impulse response with constrain after deconvolution. This curve has a lot of noise at the beginning but the later data is very smooth. We cannot observe any variation at 29 hours. It looks like a good linear system response. But it actually is a nonlinear system. The reason is that the deconvolution algorithm has forced a nonlinear response to change into a linear response. Therefore this is not the true response of real system.

The third way is to apply von Schroeter's approach, without curvature. Considering the noise in rate and pressure, the deconvolution equation can be solved with the TLS approach.

$$\Delta p + \varepsilon = C(z)(q + \delta) \tag{4-8}$$

Where $q = (q_1, \cdots, q_n)$, ε, δ are signals representing the measurement errors in pressure and rate, C is a matrix-valued function of the response coefficients with components

$$C_{ij}(z) = \int_{-\infty}^{\ln T} \theta_j (t_i - e^{\tau}) e^{Z(\tau)} d\tau \tag{4-9}$$

The TLS formulation of the deconvolution problem is to find the response coefficient z with the smallest perturbations ε and δ. Here we did not add the least overall

curvature for constraint. This leads to the optimal problem:

$$E(z) = \|\varepsilon\|_2^2 + v\|\delta\|_2^2 \tag{4-10}$$

Where v is fixed weight. This minimum problem can be solved by the approach which was introduced nonlinear total least method.

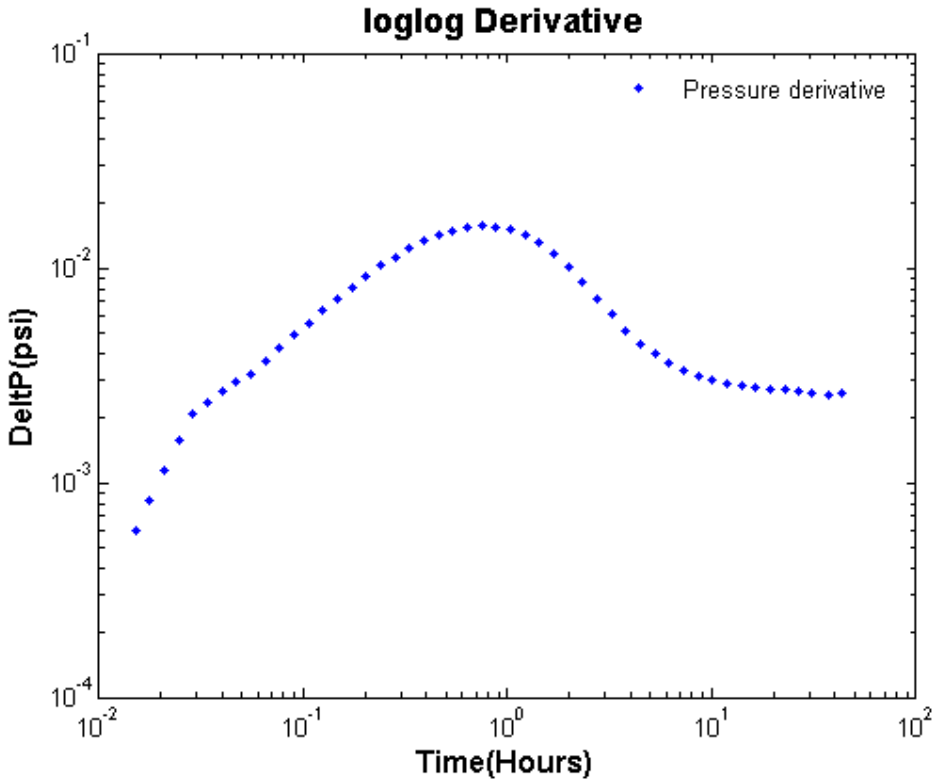


Figure 4.4 Log-log derivatives of noised nonlinear system after deconvolution

Figure 4.4 illustrates the log-log derivative from von Schroeter's approach. We find the log-log plot is very smooth because the algorithm suppresses the variation of the nonlinear system. Also, we can find neither any negative values nor any variation at 29 hours. This means the log-log plot cannot give us the correct information about the reservoir.

From these three cases, we find that the current approach deconvolve a nonlinear system into a linear system response. The reason of this problem is caused by the constrained condition of the current approach. The constrained condition of the current approach is based on the linear system. Thus the nonlinear system will be forced to be represented by a linear system. This will cause an incorrect analysis of the reservoir system. Therefore, the subsequence parts will discuss what happens when the unconstrained deconvolution approach is applied for a clean dataset.

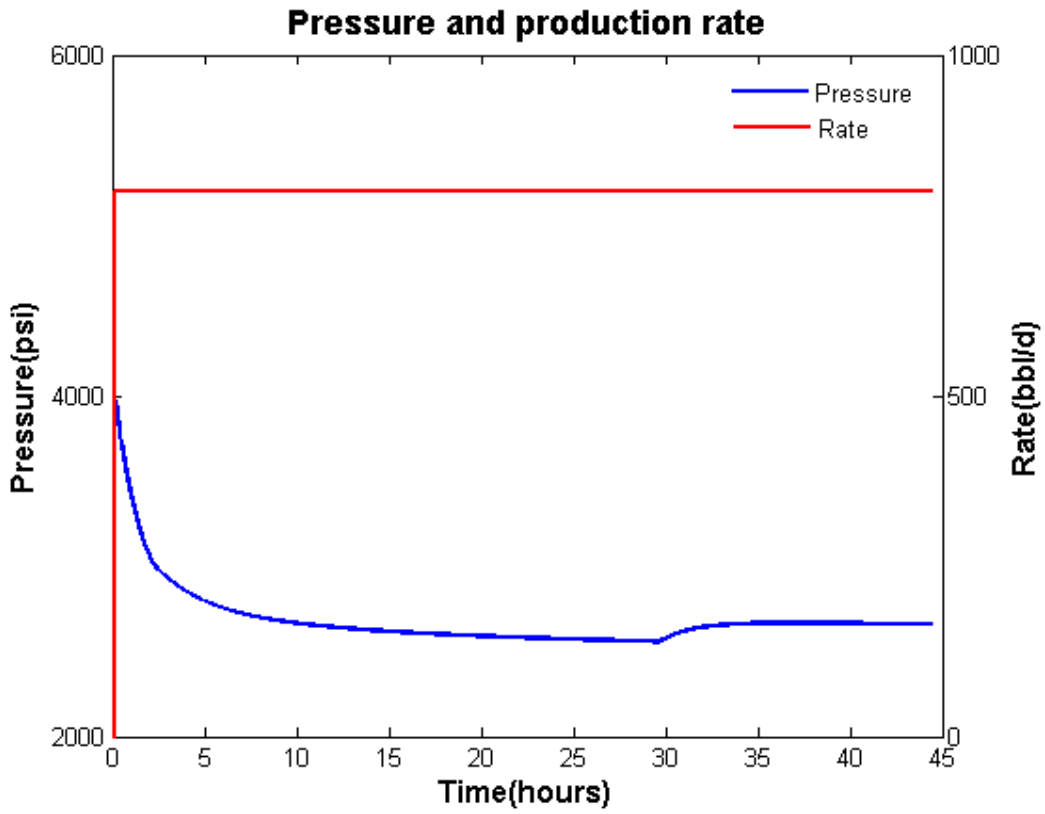


Figure 4.5 Pressure and rate of nonlinear systems with changed skin factor

Figure 4.5 shows a simulated reservoir system with changed skin factor at 29 hours. The pressure data and rate data are clean without any noise. The red line is the production history. The blue line is the pressure response for the nonlinear reservoir system. We try directly the three deconvolution approach to calculate the impulse response, unit rate response and the log-log derivative.

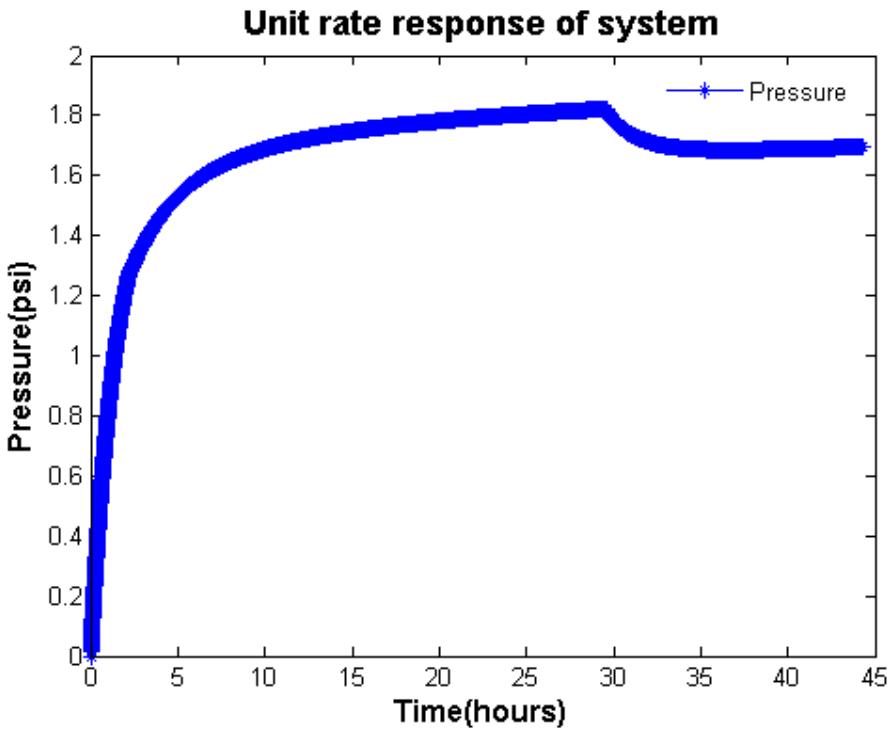


Figure 4.6 Unit rate response of nonlinear system without noise and constrain

The first deconvolution is applied to obtain the unit rate response of the nonlinear system without any constrain and noise. Figure 4.6 shows the unit rate response of the nonlinear system. We find that the curve does not meet the condition of the linear system at the 29 hour. The front part before 29hours can be considered as a linear system because the curve is positive and increasing. The second part of curve deviates from the trend of the front part. Therefore, the second part of curve is thought as the different system from the front part of curve. The 29hours point is critical point when system changes.

The second deconvolution approach is applied to obtain the impulse system response from clean pressure and rate without constrain. The impulse response of the nonlinear system is shown in Figure 4.7. As we observed, there are sharp changes at 29 hours where the skin factor changes from 5 to 4. For a linear system, the impulse response should be a smooth decline curve. The variation at 29 hours is caused by the changed skin in nonlinear system. In this way the variation in the nonlinear system can be identified from the deconvolution impulse response.

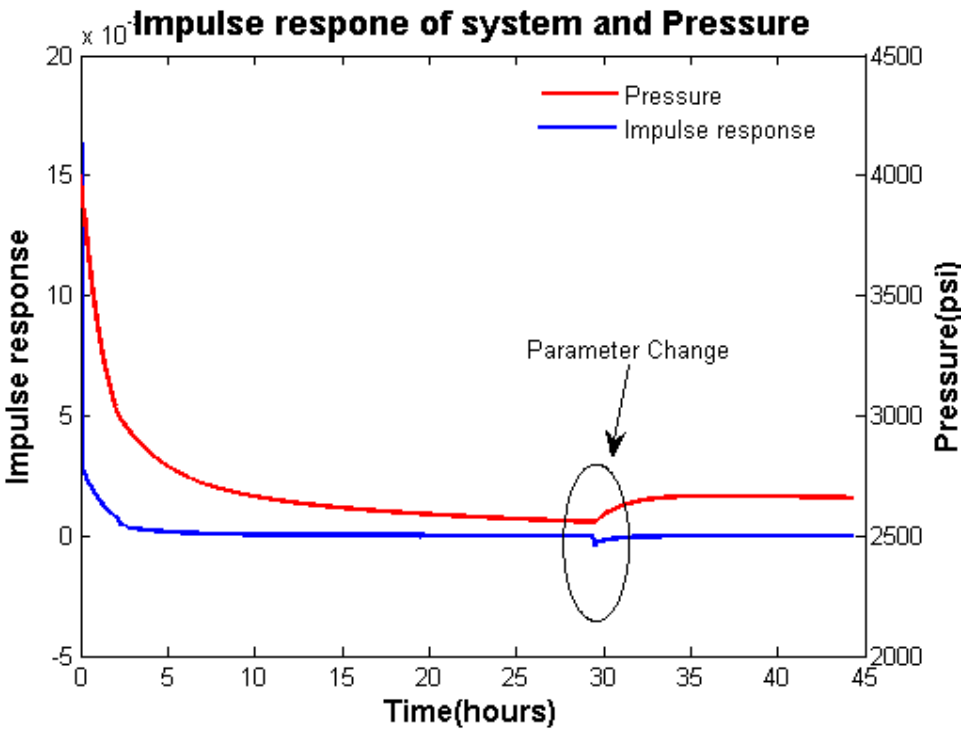


Figure 4.7 Impulse response of a nonlinear system without noise and constrain

The third deconvolution approach is direct to obtain the log-log derivative. The log-log derivative reflects the reservoir properties. And the log-log derivative is very sensitive to the variation of pressure data. For a linear reservoir system, the log-log derivative should be a positive and continuous curve for a clean dataset. The normal log-log derivative of a linear system should satisfy the following condition:

$$\gamma(c)-\varepsilon<\gamma(\mathrm{t})<\gamma(c)+\varepsilon \quad \text{With} \quad c-\delta<\mathrm{t}<c+\delta \qquad (4\text{-}11)$$

$$\gamma(t)>0$$

$\gamma(t)$ is the pressure derivative; δ, ε are very small values. Figure 4.8 is the result of the log-log derivative. We find there is a gap which has a negative value during 19 and 20 hours. This means the linear system cannot represent the current system, although the deconvolution can still be used before these points. The linear deconvolution cannot be used for the data after these data points. This log-log plot cannot be used to calculate the reservoir properties.

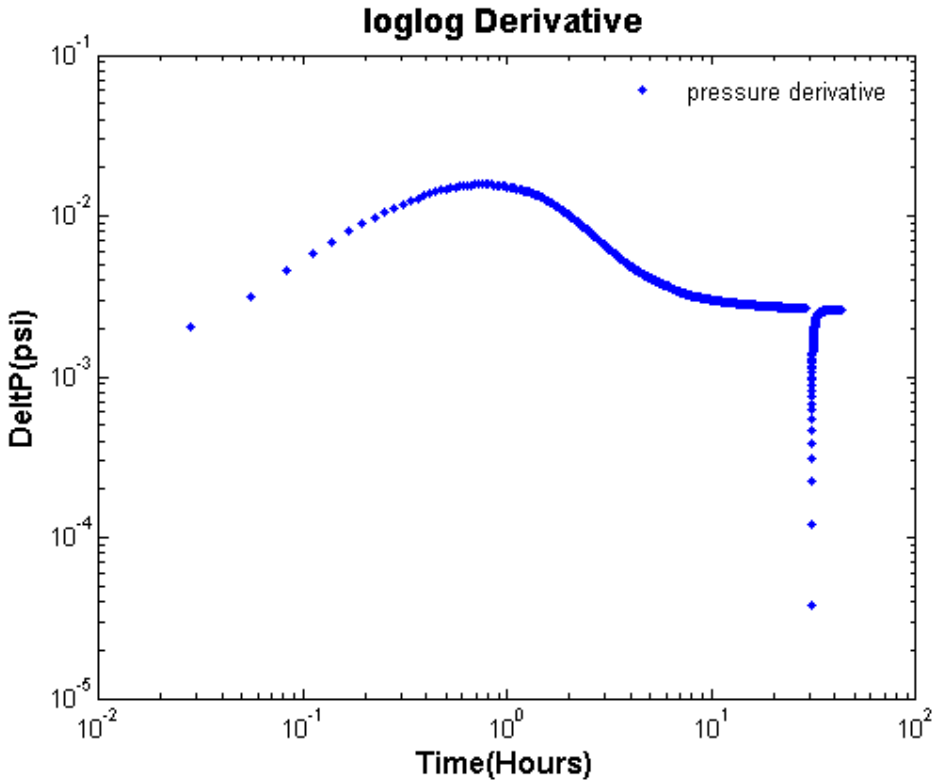


Figure 4.8 log-log plots of a nonlinear system without noise and constrain

The previous three figures show the ideal situation. The three curves show the variation when the system properties change. Therefore the deconvolution method can be applied to identify the nonlinearity of nonlinear systems. We consider using deconvolution without any constraint for noise pressure and rate.

4.2.3 Identifying the variation of nonlinear reservoir model

The idea in applying the deconvolution to perform a diagnosis is to divide the nonlinear system into a piecewise linear system. The Deconvolution method is based on linear system theory. When the deconvolution is applied to a nonlinear system, the system response causes some fluctuation as the system parameters change. From the ideal case, we found that the impulse response, unit-response and log-log derivative of nonlinear reservoir systems cause a sharp change when there are some variations from reservoir properties. But, when the unconstrained deconvolution approach is applied for the noised data, the fluctuation can not be identified in the curve.

Figure 4.9 shows the impulse response of noised pressure and rate. The impulse response is calculated with TLS and without any constrain. We can not find the fluctuation at 29hours because the nonlinear system information is flooded by the noise

in this figure. So, the deconvolution approach with constrain clean the noise and also remove the real signal of reservoir system. A new deconvolution approach is necessary to be developed to keep the useful information of reservoir while the noise also can be removed.

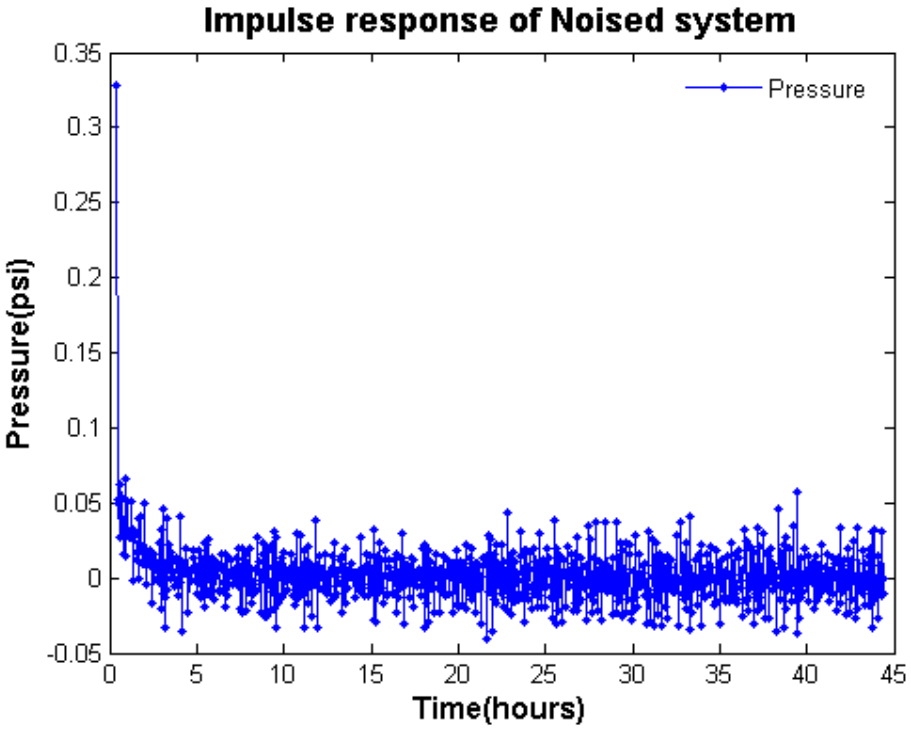


Figure 4.9 impulse responses of noised nonlinear system without constrain

In the new approach, we calculate the unit rate response from the impulse with accumulation function. The unit responses of noised nonlinear system are illustrated in Figure 4.10. We can observe the variation which is caused by the nonlinear system at 29 hours. This trend of the whole dataset is same as Figure 4.6 This means the accumulated impulse response exposes the nonlinear information of the reservoir because the unit response enlarges the variation of nonlinear information and the integration of impulse response suppress the noise of system. However, the unit response still can not represent the exact variation point of nonlinear system.

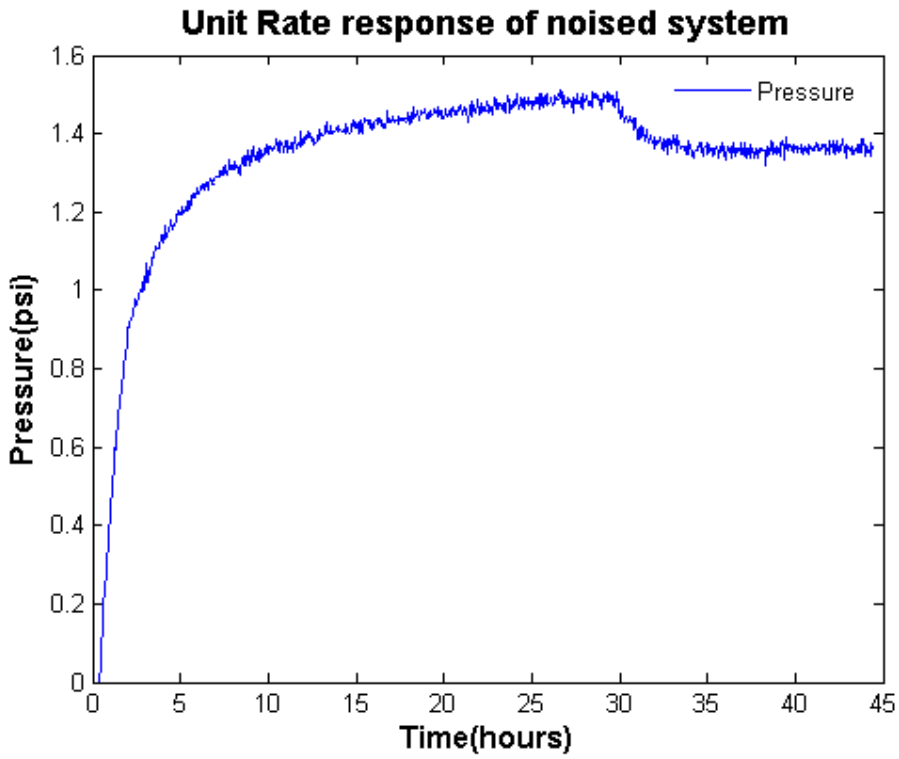


Figure 4.10 unit responses of noised nonlinear system without constrain

So, we consider calculating the log-log derivative to detect the point because the log-log derivative enlarges the small change of system. Figure 4.11 shows the log-log derivative which is calculated from the unit rate response signal after denoised unit response. We can observe a negative value at 29 hours which indicates that there is variation in the nonlinear system because the log-log derivative of linear system is continuing and smooth curve. The log-log derivative enlarges the differences in the variation of the nonlinear system, making them more obvious and useable for our purpose.

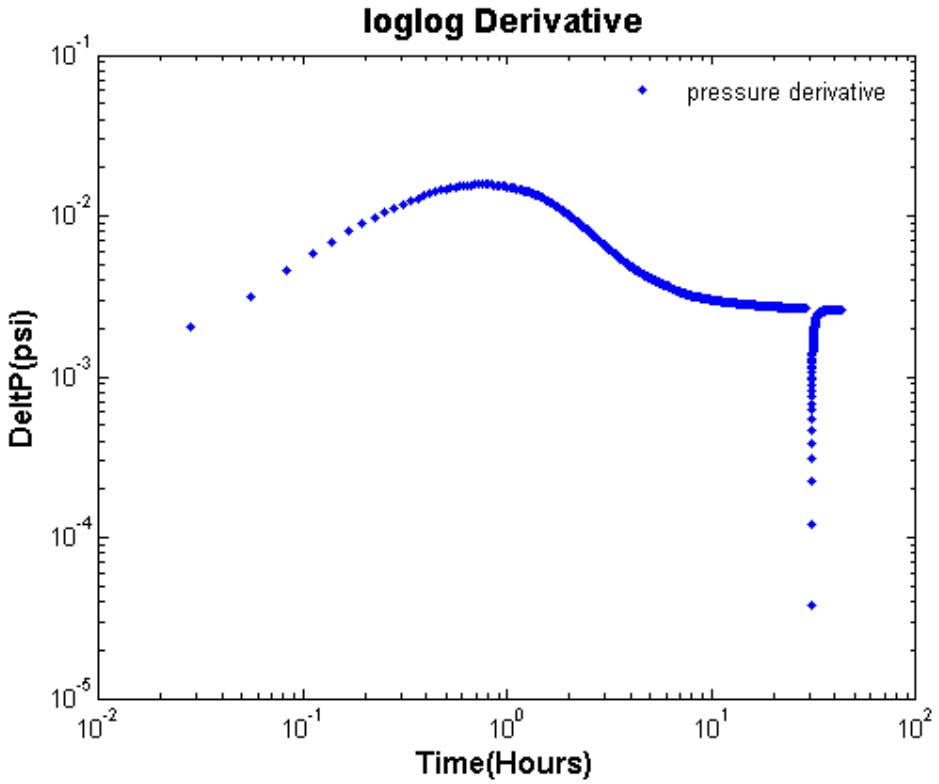


Figure 4.11 log-log plots noised nonlinear system without constrain

From the above discussion, we can draw the conclusion that the variation of a nonlinear system can be identified when the nonlinear system is caused by the variation of reservoir properties in a single well and single phase reservoir. The following three-steps approach is presented in order to identify the different linear period during the nonlinear system. The first step is to perform deconvolution to obtain the impulse response with TLS and without any constraint, considering the error of input and output. The second step is to calculate the unit response from impulse response and calculate the log-log derivative from the unit response. The derivative at any point is determined by finding a weighted mean of the slopes to a preceding point and a following point (Bourdet et. al, 1989 [59]). The final step is to find the first break point or negative point of the log-log plot data. These points can be considered as the boundary of the first linear system. After identifying the first linear system, the consequent data are applied to repeat the last three steps to detect the next linear system. The data at the beginning of the next linear system can be the last data of the previous linear system.

4.3 Case study

4.3.1 workflow

A new workflow of dynamic forward modeling comprises three steps. The first step is to apply traditional well testing for the selected BU in order to obtain the changed parameters. This result gives the first estimate of the location of which period is a linear system. The second step uses deconvolution to refine the range of the linear system. After the linearization of the nonlinear system with deconvolution, the nonlinear system is divided into separate linear systems. The numerical well testing approach is applied for the first linear system in order to achieve the long term and short term characteristics from the PDG pressure data. The system response of the first linear system is achieved from the deconvolution during the first linear system period. The log-log plot of the system response can help numerical well testing to obtain the transient information about the reservoir, while the Cartesian plot of PDG pressure data can reflect the long term character of the system. After matching the PDG pressure in a different plot in order to validate the reservoir model, the numerical reservoir model is confirmed with certain criterion. The subsequent linear systems are repeated with the forward numerical well testing steps. After finishing the last linear system, the whole nonlinear reservoir system is dynamically updated with the integration of numerical well testing, deconvolution and dynamic well testing. The following shows a synthetic case of this workflow.

4.3.2 Traditional well testing

This synthetic nonlinear case study is simulated with ECLIPSE. The PDG pressure data simulated for more than six months and includes 50,000 points. The reservoir model is a single phase, homogeneous and closed system. In order to demonstrate the dynamic reservoir system, the reservoir properties like the skin factor and permeability is changed with time. Figure 4.12 shows the simulated PDG pressure.

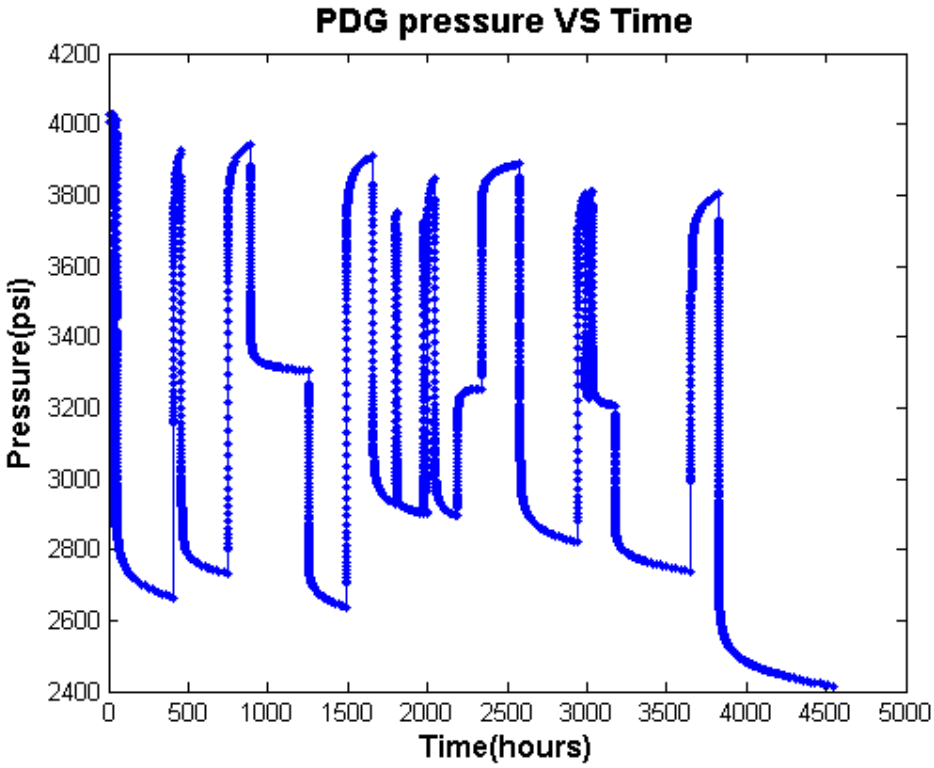


Figure 4.12 PDG pressure data of Synthetic nonlinear reservoir

Before analyzing well testing data, the PDG data need to be processed with the Wavelet approach. This approach identifies the flow event to distinguish the whole PDG data into different piecewise BU or DD. Further processing is applied for a selected shut-in BU period. Table 4.1 shows the detail of the identified flow period. The first and second column shows the beginning and end of every flow period. The third is how long every flow period produced. The fourth column shows the flow type (1 means DD; 0 means shut-in BU; and -1 means rate-drop BU). The whole PDG dataset contains 20 DD flow periods, 11 shut-in BUs and 1 rate-drop BU. Then the denoised approach is applied to get the clean PDG pressure. Figure 4.13 shows the separated flow period in a different color.

Begin(hours)	End(hours)	Flow period(hours)	Flow Type
0	24	24	0
24	26.4	2.4	1
26.4	28.8	2.4	1
28.8	31.2	2.4	1
31.2	48	16.8	0
48	50.4	2.4	1
50.4	410.4	360	1
410.4	458.4	48	0
458.4	746.4	288	1
746.4	890.4	144	0
890.4	1250.4	360	1
1250.4	1490.4	240	1
1490.4	1658.4	168	0
1658.4	1802.4	144	1
1802.4	1804.7999	2.3999	0
1804.7999	1972.7999	168	1
1972.7999	1975.2001	2.4002	0
1975.2001	1999.2001	24	1
1999.2001	2047.2001	48	0
2047.2001	2191.2001	144	1
2191.2001	2335.2001	144	-1
2335.2001	2575.2001	240	0
2575.2001	2577.6	2.3999	1
2577.6	2580	2.4	1
2580	2582.4	2.4	1
2582.4	2942.4	360	1
2942.4	2990.4	48	0
2990.4	3007.2001	16.8001	1
3007.2001	3031.2001	24	0
3031.2001	3175.2001	144	1
3175.2001	3655.2001	480	1
3655.2001	3823.2001	168	0
3823.2001	4513.4626	690.2625	1

Table 4.1 Identified BU and DD of nonlinear reservoir

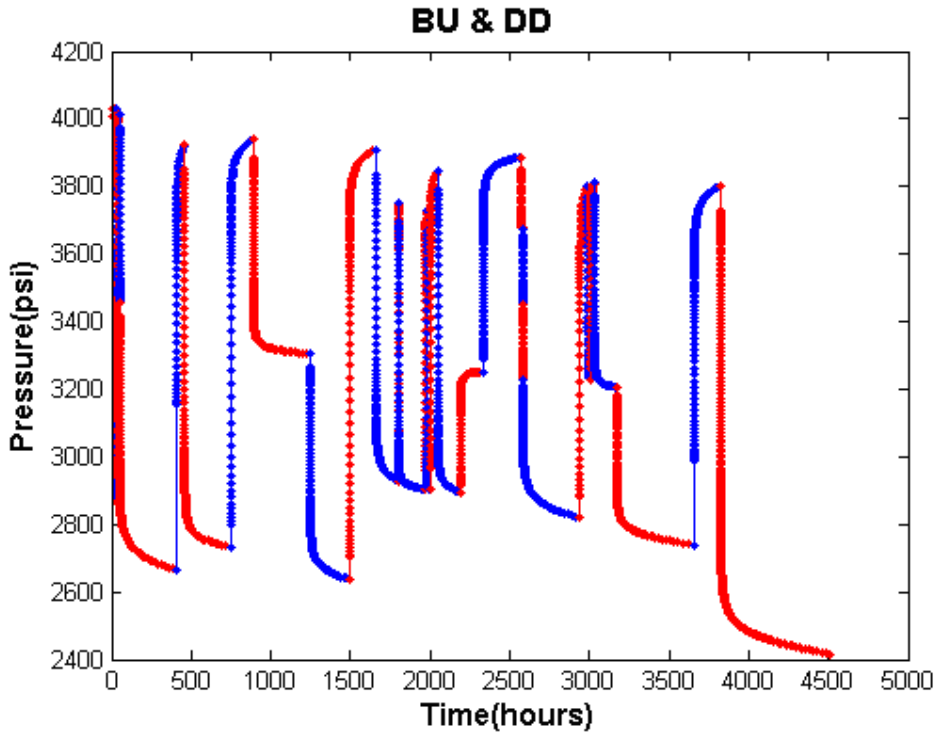


Figure 4.13 Separated BU and DD nonlinear reservoir

After processing the PDG data with wavelet transform, the longest shut-in BU is selected to perform well test analysis. The normal well testing steps are conducted for this BU. The engineer decides the reservoir model and the flow regime from the log-log plot according to his or her experience. From Figure 4.14, a radial flow and close reservoir system is selected as a model. The early, middle and later flow regime is defined. Then the specialist analysis is operated to obtain the permeability and skin in the semi-log plot. Figure 4.15 shows the result from the semi-log plot. All this information, including the reservoir model and the definition of flow regime will be saved as the template for all other BUs.

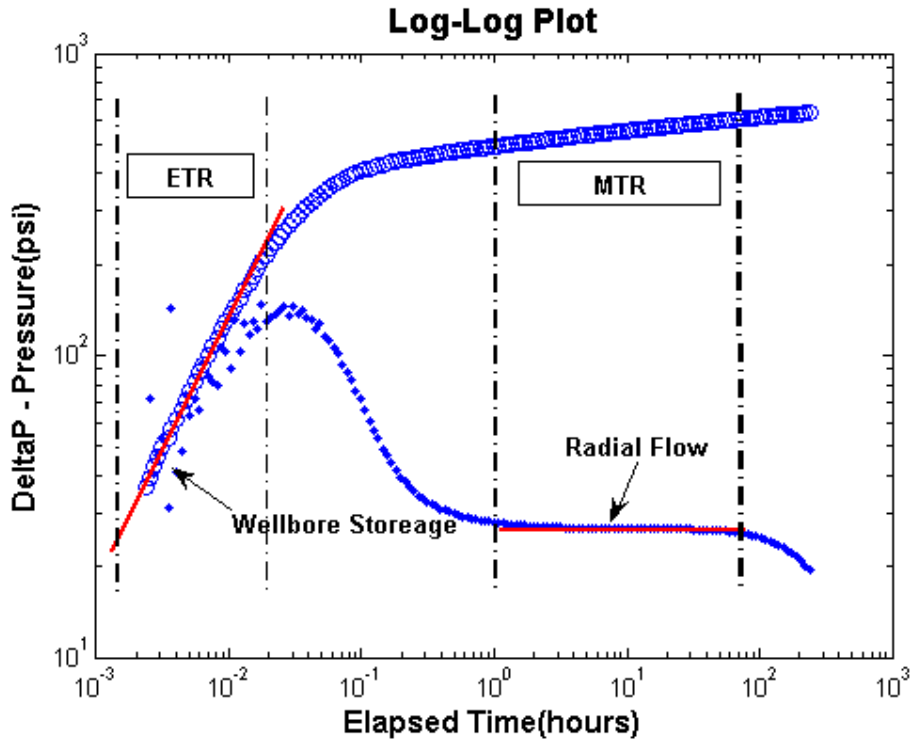


Figure 4.14 Log-log plot of longest BU in nonlinear reservoir

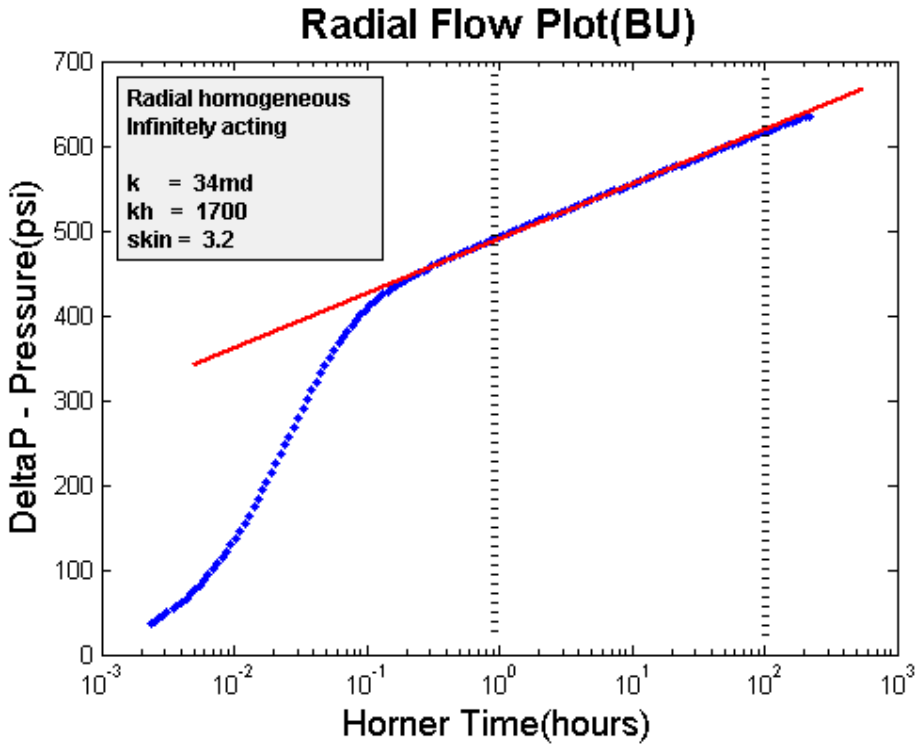


Figure 4.15 Semi-log plot of longest BU in nonlinear system

After defining the template for all BUs, the subsequent BUs will be analyzed automatically with the template of the first BU. Figure 4.16 shows the multi-semi-log.

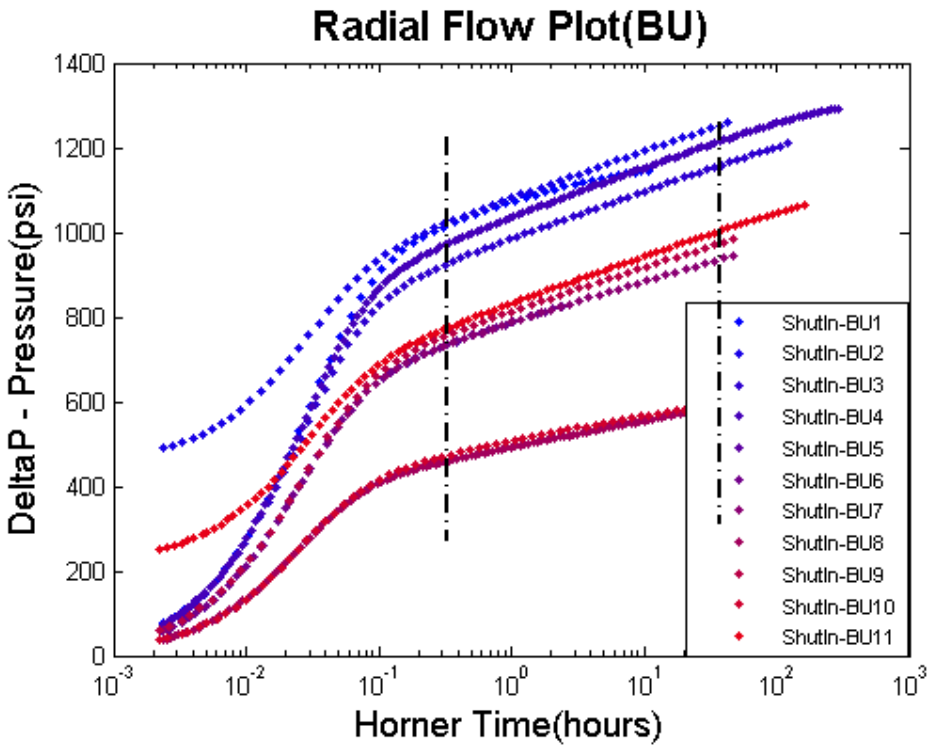
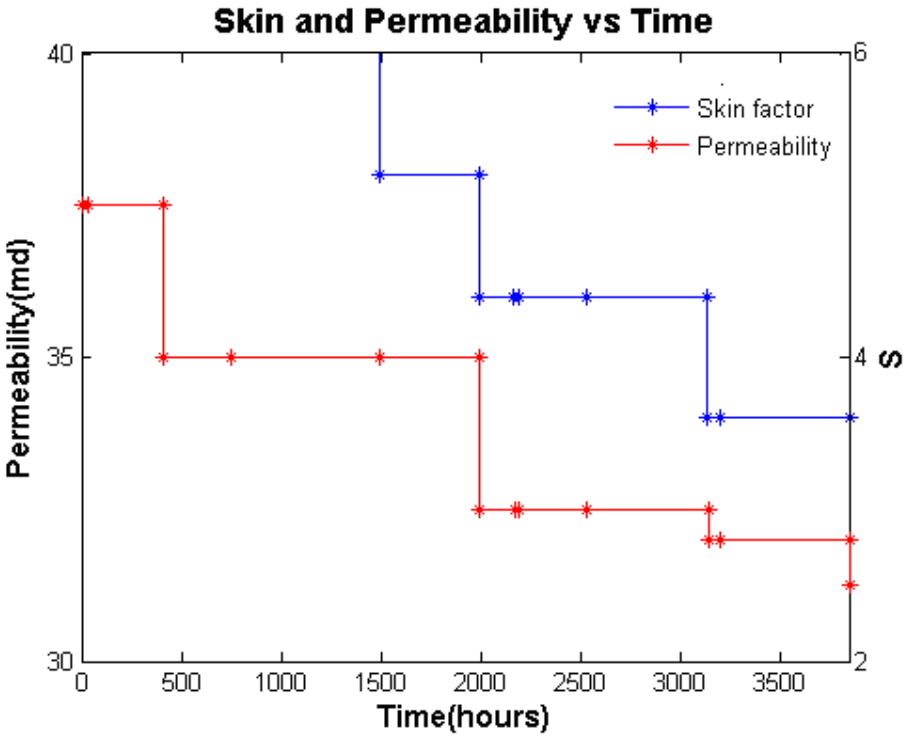


Figure 4.16 Semi-log plot of all BUs



Fi

Figure 4.17 Changed parameter from all BU

Figure 4.17 shows the changed skin factor and permeability from the semi-log plot. But, it is very difficult to apply the current result for the dynamic reservoir model because the result is only part of the information from the long term data. We do not

know whether the parameters change during the DD period. Hence it is necessary to identify the range of every parameter.

4.3.3 Identifying linear system

In this case study we will apply the three steps approach to identify the range of a constant linear system. Before the deconvolution approach is applied for identification, the first calculation of separation of the nonlinear system can be estimated from the changed parameters in Figure 4.17. In Figure 4.18, we can divide the original PDG data into five different parts according the constant value of skin and permeability. For example, the first part can be defined from 0 hours to 380 hours because the permeability and skin factor are constant.

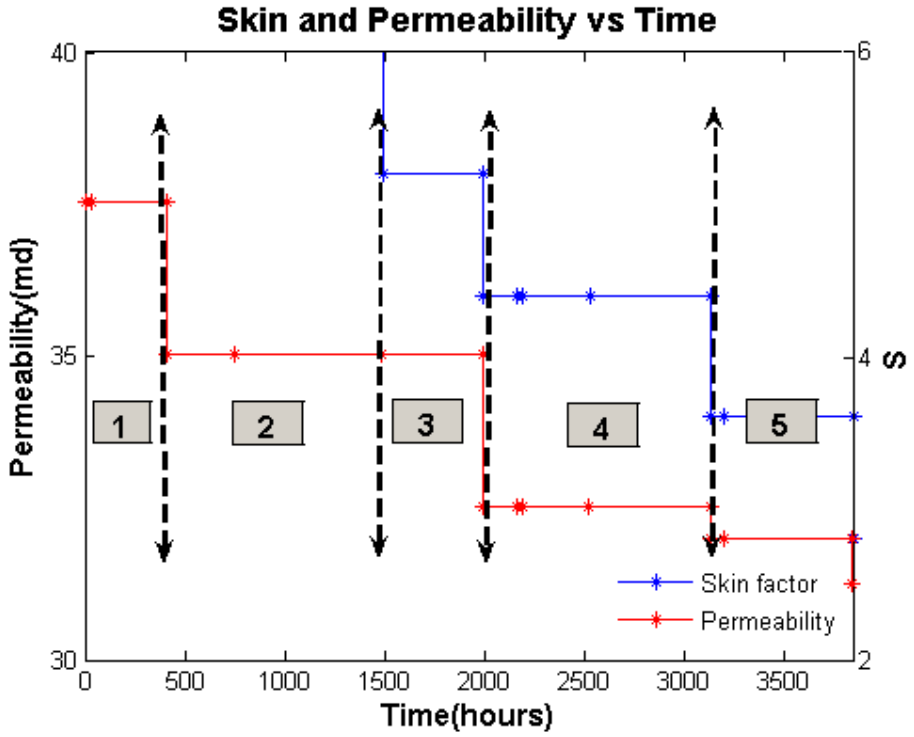


Figure 4.18 Different constant system periods

The first estimate indicates that the end of the first linear system may be in 386.4 hours. So we can simply use part of the PDG data to identify the first linear system. This can be more efficient than using all of the PDG data. If the algorithm can identify the boundary of the first linear system during part of the PDG data, the algorithm goes to the next part to identify the second boundary of the linear system. If the algorithm cannot identify the boundary of the first linear system, then it will extend the searching period of the PDG data which means using more PDG data until the algorithm search

reaches the boundary of the linear system.

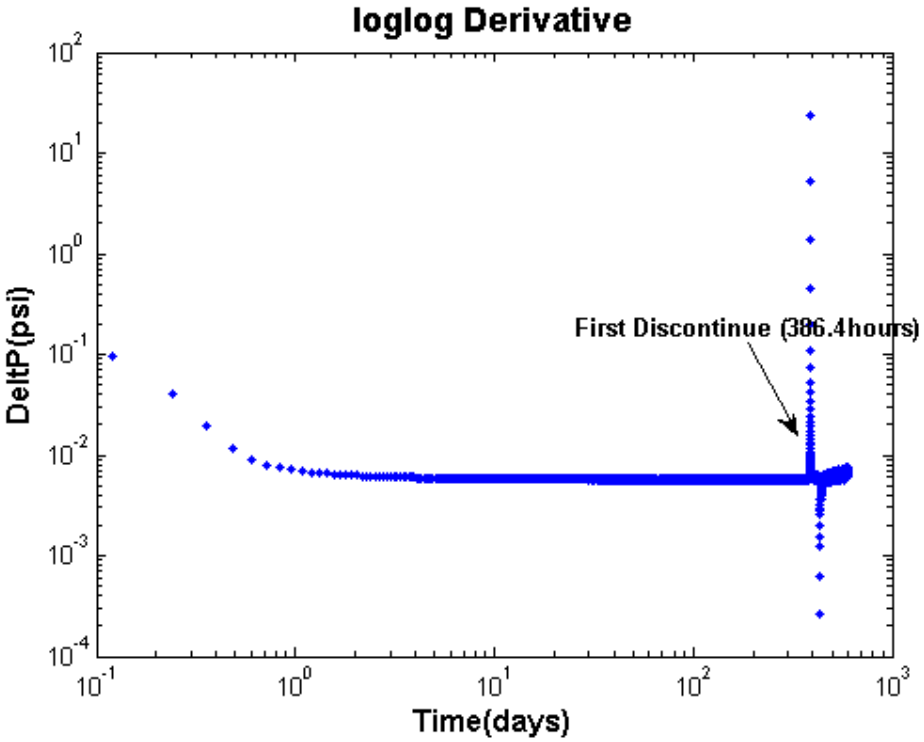


Figure 4.19 Searching for the first boundary with 600 hours data

Figure 4.19 uses 600 hours worth of data to identify the location of the first boundary. In this example we find there is a first discontinuous part at 386.4 hours. This discontinuous signal means the previous period can be considered as a linear system. We also find some discontinuous signals after 386.4 hours because the first discontinuous signal is accumulated when the rate changes. Hence we need repeat the same steps to identify the boundary of the second linear system from the following PDG data, until the end of the PDG data is reached. After the identification of the linear systems, the original nonlinear system can be divided into six linear systems. Table 4.2 shows the result of the identification of the nonlinear system.

Linear system Number	Begin(hours)	End(hours)
1	24	386.4
2	386.4	890.4
3	890.4	1658.4
4	1658.4	2575.2
5	2575.2	3175.2
6	3175.2	4513.4626

Table 4.2 Separation of nonlinear systems

4.3.4 Dynamic numerical well testing

The normal procedure of history matching considers the reservoir model as a static model in which the reservoir parameter does not change with time. In Figure 4.20, the blue line is the real PDG data and the red line is simulated data with normal procedure. These two lines do not match because we only consider the dynamic reservoir as a static model.

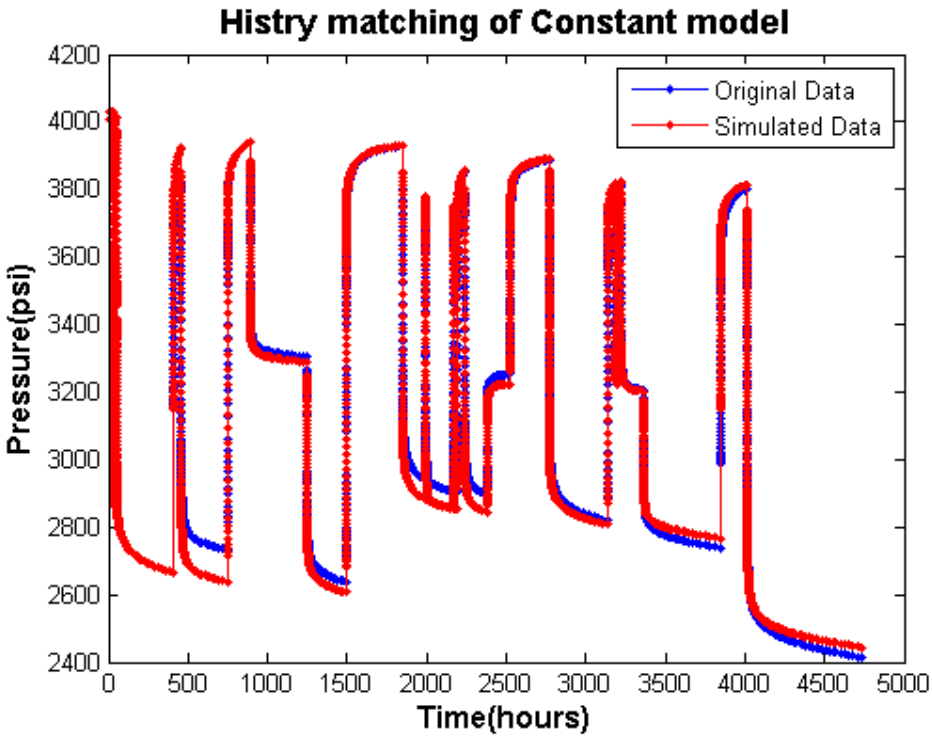


Figure 4.20 Traditional history matching

Since the reservoir model is a dynamic nonlinear model, the nonlinear system can be divided into different linear systems with the help of deconvolution. From Table 4.2, the whole reservoir system can be looked as six linear systems. For every linear system period, we can carry out numerical well testing to validate the reservoir model, because deconvolution can transform these data as a constant rate response. Figure 4.21 shows the log-log plot history matching of the first linear system. The log-log plot is the response of a constant rate of 1000stb/d. The history matching is started from a initial point and then the error is considered as the reference for the direction and timestep of a new test.

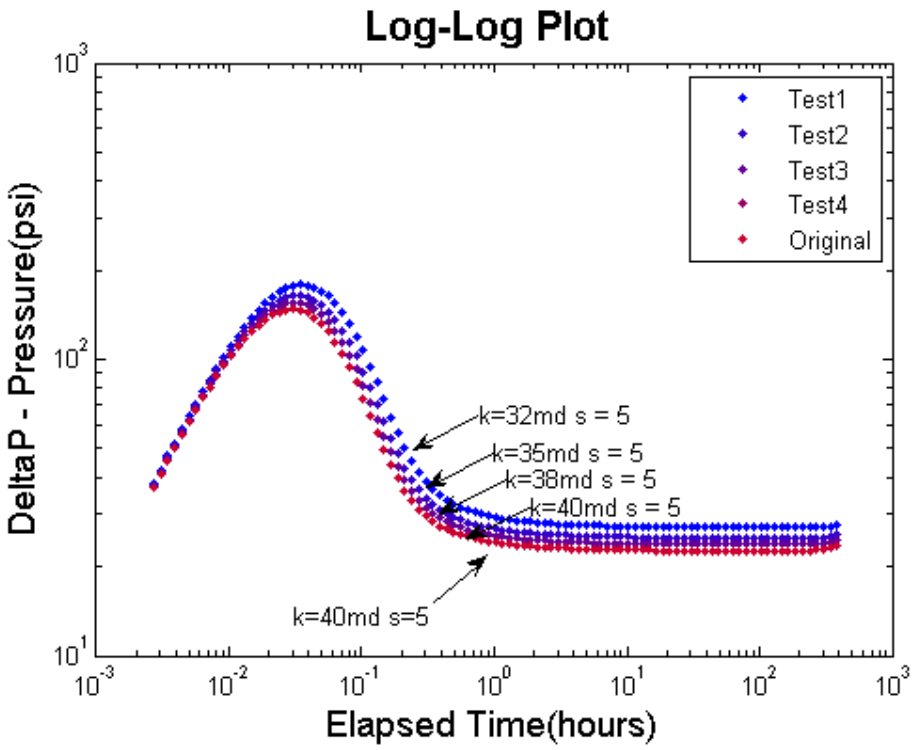


Figure 4.21 History matching of first linear system

The following five linear systems are also subjected to numerical well testing to validate the reservoir model. The only difference is the start state of the subsequent linear reservoir models. Every reservoir model restarts from the last point of the previous reservoir model. Figure 4.22 shows that history matching result of the whole system is much better than that in figure 4.20.

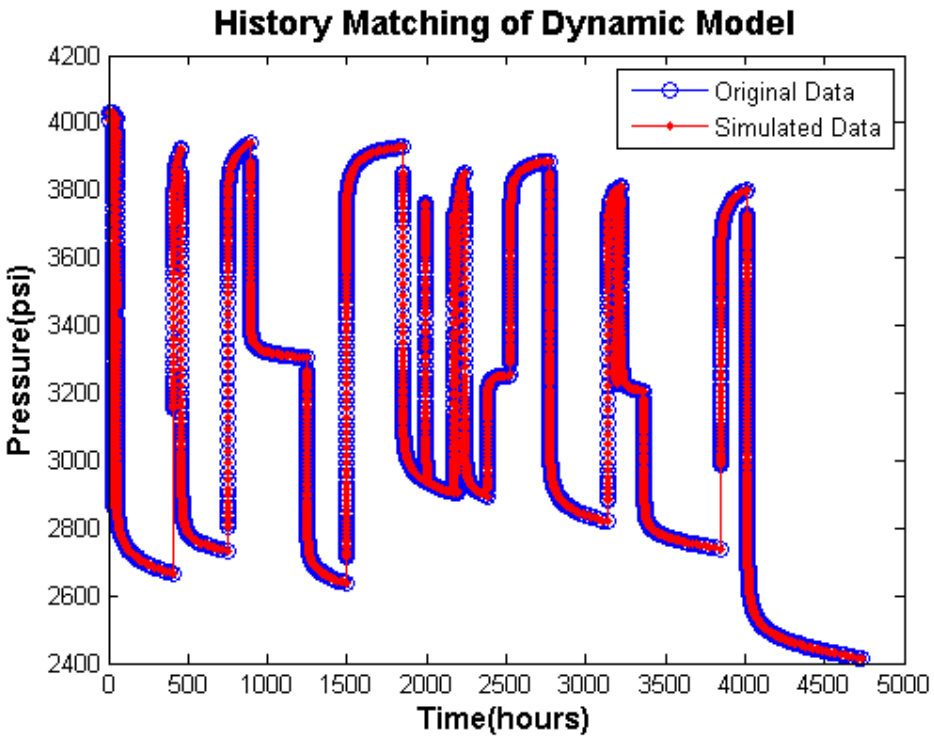


Figure 4.22 History matching of the whole system with dynamic parameter

4.4 Chapter conclusion

This chapter has carried out an analysis of PDG pressure data using dynamic forward modeling with the help of deconvolution. This approach considers the PDG pressure as a response from nonlinear reservoir system. The nonlinear system is divided into separated linear systems by the deconvolution system. The normal deconvolution approach is used to transfer the multi-rate problem to a constant rate problem with some constrain for linear system, but these constraining factors may force a nonlinear system to be considered as a linear system. However, when a nonlinear system data is calculated with deconvolution, the result of deconvolution will show discontinuous or negative values at the points of change. This phenomenon is applied to diagnose the linear systems within the nonlinear system. The separated linear systems can then be analyzed with numerical well testing. The whole nonlinear system can be matched with moving window analysis approach. A three-step approach is then used to analyze the PDG pressure:

The traditional well testing approach is first applied to analyze the PDG data. The changed reservoir properties can be required from the BU of PDG data. This is the first guessed location of linear system. The second step is to diagnose linear system.

The new approach considers the errors of both input and output without any constraining in deconvolution. Deconvolution is used to to obtain the impulse response with TLS. Then, the unit response and log-log derivative are calculated from impulse response from the unit response. The final step is to find the first break point or negative point of log-log plot data. These points can be considered as the boundary of the first linear system. So, a nonlinear system can be divided into several linear systems. The dynamic numerical well testing method is operated for each linear system period to validate the reservoir model. Each linear period is restarted from the last point of the previous linear period. The whole set of data will be analyzed with this dynamic forward modeling procedure.

CHAPTER 5 CONCLUSIONS AND FUTURE WORK

The first part of this chapter will draw conclusions from the area of PDG pressure data processing and analysis. The advantages and limitations of our approaches are discussed in this part. The second part of this chapter will propose a range future work of based on our current experience.

5.1 Conclusions

This thesis has put forward a systematic workflow to utilize the PDG pressure information from the view point of well testing. This workflow comprises two parts. The first part is to perform data processing for PDG data with the help of wavelet transforms.

PDG pressure data represent a very complex convolution of information from the downhole well environment because it is recorded under natural conditions that can not avoid the effect from gauge, well, reservoir and other factors. Therefore, the target of PDG data processing is to filter different kinds of information from PDG pressure data in order to receive more clean and simple information.

Our idea of data processing is experiential and comes from the observation and of huge amounts of different real reservoir PDG data. The PDG pressure information is classified into three kinds of information: gauge information, well information and reservoir information. From our experience, the gauge information includes some outliers, negative value, data gaps, repetitive values and so on. The outliers are often found in some early stage PDG data. The well information consists of opening a well, shut-in a well, well work-over and the well inflow. The reservoir information is affected by the skin factor, reservoir parameters, reservoir boundaries and the effect from other wells. We also found gauge information and well information located in the high frequency signal, while the reservoir information is represented in the low frequency signal. This is why we choose the wavelet approach to do data processing.

However, the traditional wavelet approach, called the modular maximum approach, can only treat ideal PDG pressure data. The limitation of this approach is that some real PDG pressure data can not find the local modular maximum.

Hence, in our approach, we use the wavelet transform to divide the original PDG pressure signal into different frequency information. We found that gauge information, well information and reservoir information have different characteristics in different frequencies. These characteristics were than used to identify a wide range of reservoir details. This approach has been tested by an oil service company and a software company in North Sea oil reservoir and gas reservoir PDG data. Different frequency PDG pressure data has also been processed with approach. This approach has proven to be an efficient method to process real PDG pressure data and a service company has already implemented this method. It should be noted, however that in some cases, our approach may not work well because of some pathological situation not covered in this algorithm.

In outlier removal, we have improved the traditional outlier removal approach. Our approach not only removes spike outliers but also removes step outliers. However, we found it is very difficult to remove the outlier when the outlier event and flow event happened at the same position because we can not distinguish two kinds of events.

After outlier removal, our algorithm focused on well event identification. The flow event identification algorithm is based on the wavelet method. Flow event identification is the key step of processing PDG pressure. The new algorithm is more suitable for real PDG data and can not only identify a large flow period but also detect a small flow period. Also, the algorithm considers the abnormal response of wellbore storage. During the test of the algorithm, we found it to be very sensitive to the threshold which is used to group the flow event data in high frequency. So, we designed a more flexible algorithm which can automatically choose the threshold from of data.

After flow event identification, three conditions are considered to distinguish the BU and DD of every flow period. The temperature is found to hold important information to distinguish the BU and DD. The potential of temperature has not been paid much attention in PDG data processing. In addition, we discovered some interesting behavior of well and reservoir in PDG pressure data. The well interference, which is always considered as abnormal behavior in other articles, can be detected by this new algorithm. However, there are still some unknown behaviors of reservoir and well in PDG pressure data. Further processing part including denoise and data reduction is required to denoise a single flow period is better than to denoise the whole flow period with the wavelet method because the smearing effect at the beginning and end of every

flow period can be avoided. The data reduction algorithm aimed to sample the original data in log scale for saving memory and improving the calculation speed during simulation.

In the second stage of the thesis, data analysis was carried out for PDG data, and was integrated with current approaches: dynamical well testing, deconvolution and numerical well testing. The first issue of PDG data analysis was to recover production history from PDG pressure and cumulative production with the newly developed approach. This new approach allocated the cumulative production rate for every flow period according to the amplitude of the high frequency PDG pressure signal located at the beginning of every flow period. This approach gave a perfect result for oil reservoirs to recover production history from PDG pressure and daily rate. The new method was then extended to recover production for grouped wells without a shut-in. This recovery rate approach was based on the signal and linear systems theory which assumes the reservoir system is a linear system for every short period We found this new approach gave a good performance in oil reservoirs with small wellbore storage or smart well.

The next issue of PDG data analysis was to apply the traditional well testing and numerical well testing to analyze the PDG pressure data. The traditional well testing was applied for every BU of PDG pressure to get modified reservoir parameter (eg, skin factor, permeability, and boundary). In general, the traditional approach is inadequate because only the BU is analyzed to get the parameter. So, this changed parameter can only give a guide of reservoir. Another way to analyze PDG pressure is to use numerical well testing. We developed a new toolbox to realize numerical well testing for PDG data that can be coupled to a simulator like eclipse.

The third issue of data analysis considered here was to analyze nonlinear PDG pressure data. The idea of analyzing the nonlinear PDG pressure is to linearize the nonlinear system into separate linear systems. The new approach made use of deconvolution to diagnose the critical points of linear systems in nonlinear PDG pressure data. Then every linear system was analyzed with the numerical well testing individually. The deconvolution approach is normally used to transfer the multi-rate problem to a constant rate problem with some constraints, but this constraint may force a nonlinear system into a linear system when the assumption of deconvolution is neglected. In our approach, we used the characters of deconvolution approach, where deconvolution shows discontinuous or negative values at the crossover points when there are sudden

changes in the nonlinear system. This approach worked very well in our synthetic case, where the skin factor and reservoir permeability changed with time. But this approach also has its limitations. In our reservoir case, we just considered one kind of nonlinear system, caused by change in reservoir parameter. But in the real world, nonlinearity system may be caused by multiphase, multi-well phenomena amongst other things.

In line with our workflow, a toolbox called PRIME_TOOLBOX has been developed with Matlab to facilitate PDG pressure data processing and analysis. The data processing part of the toolbox shows good performance and has been tested on huge amounts of real PDG pressure data. The work presented in this thesis forms the basis of a new, cost-effective approach to dynamic well testing.

5.2 Future work

In the future, reservoirs can monitor information like pressure, rate and temperature in real time. This real time information will reveal the real situation of the reservoir and can help reservoir managers to gain real time control and optimize the reservoir's performance. The future work based on this thesis would mainly suggest two directions: data processing and data analysis.

5.2.1 Future work on data processing

The current data processing algorithm can process pressure data, including PDG pressure data, DST data and mini-DST data. The future work of data processing should extend the current processing toolbox to process temperature and production information.

The production data may be obtained from the permanent meter data, tube-line data, sparse day, and week or month datasets. The wavelet method can also be applied to these data and the production data can be used to check the pressure data. The third type of data is temperature data which is recorded from PDG or DTS systems. The PDG pressure can also be processed with the wavelet method. And the DTS data can be analyzed from a two dimensional view of the data. This data helps people to understand the real situation of the well. In addition, the current approach only considers how to process offline data, so we should also consider handling online data and batched data from PDG based on the current approach.

The data processing stage mainly involves detecting the phenomena of the well and

reservoir from PDG data. Not only are the normal BU and DD recorded, but also some new phenomena are recorded. The real time data can record the effect of the well, the influence of mechanical operation, the interface from other wells and so on. We need to detect more phenomena from the PDG data in future and the more experience we have, the more information we can identify from PDG data.

5.2.2 Future work on data analysis

The current rate recovery approach can obtain the rate history in ideal situations. This approach can be used approximately for a small wellbore storage case. However, the real situation can not avoid wellbore storage. It is necessary to extend the current approach to the real situation. There are two directions for future work. The first direction is to consider the rate variation as a multi step signal at the beginning of a flow period. The second direction is to build a model which integrates the reservoir and wellbore.

In addition, the studies presented here have only investigated one type of nonlinear system with the deconvolution approach. But the real nonlinear system can consist of hundreds of types. So, it is essential to extend the current approach to other types of nonlinear system. Since the reservoir is a nonlinear system, the future of the reservoir also changes with time. However, the reservoir PDG data using the current approach in this study is matched with flow period by flow period forward modeling. At the end of matching, we can match the whole history data set with a dynamic reservoir model. This last reservoir model can be considered as the current model which inherits all the historic and current information. This current model will be applied to forecast the future performance of reservoir. However, the problem is that the reservoir model is a dynamic model. This means we cannot use the current static model to forecast the future performance because the reservoir parameters may have been changed. The key point when trying to forecast future performance with the numerical model is to know both the current model and the parameter changes of the reservoir. There is no way to know the changed parameters of the next step. But we can discover the tendency of future performance from the black box model. The real time long term PDG pressure data is the ideal data information with which to build the neural network model and forecast future performance. This suggests that the future work of data analysis is to use numerical well testing assisted by the neural network model.

APPENDIX SOURCE CODE

PRIME_TOOLBOX is designed to process and analyze the PDG data with Matlab. This toolbox mainly included five sub-modules: data collection, data processing, traditional well testing, numerical well testing and dynamic forward modeling. The first module is involved in collecting all data that will be used in the next several modules. The data processing part makes use of wavelet transform to pre-process the PDG data in order to analyze it for the following part. This part will first remove the outlier, then identify the BU and DD and recover the production rate from PDG at last. The third part analysis with the normal well testing method after the data compression and denoise. The changed reservoir properties are exported and save to the project. The fourth part analyze the pressure data in the other way which don't break the PDG data into pieces. Numerical well testing is used to analyze the PDG data. The last part called the dynamic forward modeling which conduct the numerical well testing and deconvolution which include all information from the front part. The changed properties of reservoir from traditional method are the first guess for numerical model. The deconvolution approach is applied to diagnose the linear system from nonlinear reservoir system. And the numerical well testing is applied for every linear part.

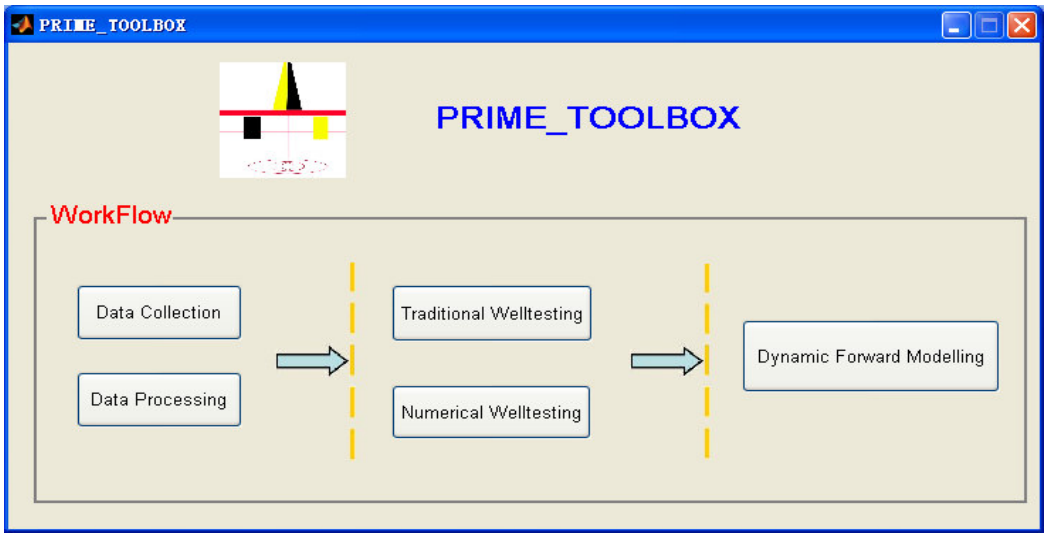


Figure Appendix.1 Main Interface of Toolbox

Data collection code

Main function

```
% DATA_COLLECTION M-file for data_collection.fig
function varargout = data_collection(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',mfilename,'gui_Singleton',gui_Singleton, ...
                   'gui_OpeningFcn', @data_evaluation_OpeningFcn, ...
                   'gui_OutputFcn', @data_evaluation_OutputFcn, ...
                   'gui_LayoutFcn', [] ,'gui_Callback',[]);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% ------------------------------------------------------------------
% --- Executes just before data_evaluation is made visible.
Function data_evaluation_OpeningFcn (hObject, eventdata, handles, varargin)
% 0. Initial the global parameters for software
% this parameter classe the different plot operation;
% plot-operation = 0; default state plot-operation = 1; click the View_Button event
%plot-operation = 2; click the plot_Button event
handles.flag_plotoperation = 0;
% this parameter classe the different plot operation;
% plot-operation = 0; don't plot figure    plot-operation = 1; plot the figure
handles.flag_graphs = 0;
handles.plotline = 0;
%%%--- create a global variety to contain data
currentwell = wellclass(); handles.CurrentWell = currentwell;
%%%create an array to contain the well data
handles.WellArray = {};
%%%%% 1. initial the menu
```

```
set(handles.menu_reservoir_data,'Visible','off');
set(handles.menu_welldata,'Visible','off');
set(handles.menu_fluid_rock_data,'Visible','on');
set(handles.menu_geological_data,'Visible','off');
%%%%% 2. initial the window and the axis
set(handles.axes_well_plot,'Visible','off');
set(handles.uipanel_left_control_well,'Visible','off');
set(handles.uipanel_bottom_state_well,'Visible','off');
%% set the second axes for axes_well_plot
axes(handles.axes_well_plot);        ax1 = gca;
ax2 = axes('Position',get(ax1,'Position'),...
'XAxisLocation','top','YAxisLocation','right','Color','none','XColor','k','YColor','k');
handles.axes2_well_plot = ax2;
set(handles.axes_well_plot,'Visible','off');
set(handles.axes2_well_plot,'Visible','off');
handles.FileName = ''; handles.PathName = '';
handles.output = hObject;       guidata(hObject, handles);


% --------------------------------------------------------------------
%                          1.FUNCTION FOR SAVE AND OPEN DATA
function menu_creat_new_project_Callback(hObject, eventdata, handles)
%0. set window as off
set(handles.axes_well_plot,'Visible','off');
set(get(handles.axes_well_plot,'Children'),'Visible','off');
set(handles.axes2_well_plot,'Visible','off');
set(get(handles.axes2_well_plot,'Children'),'Visible','off');
%%%set the state of left_control window 'off'
h = findobj('Tag','uipanel_left_control_well'); set(h,'Visible','off');
hchildren = get(h,'Children');
for(i=1:1:length(hchildren))          set(hchildren(i),'Visible','off');          end
%%%set the state of bottom_panel window 'off'
h = findobj('Tag','uipanel_bottom_state_well');      set(h,'Visible','off');
hchildren = get(h,'Children');
for(i=1:1:length(hchildren))        set(hchildren(i),'Visible','off');          end
```

```
h = findobj('Tag','menu_analysis_data');
set(h,'Visible','on'); set(h,'Enable','on');
%1. create a new file_dialog
[FileName,PathName,FilterIndex] = uiputfile({'*.well','Well file(*.well)';'*.*','All ..
file(*.*)'});
handles.FileName = FileName;      handles.PathName = PathName;
handles.FilterIndex = FilterIndex;
%2. create new different data files
if isequal(FileName,0) | isequal(PathName,0)
    disp('User selected Cancel');
else
        %a. set all menu unvisible and unable
        h = findobj('Tag','menu_reservoir_data');
        set(h,'Enable','off');            set(h,'Visible','off');
        h = findobj('Tag','menu_welldata');
        set(h,'Enable','off');            set(h,'Visible','off');
        h = findobj('Tag','menu_fluid_rock_data');
        set(h,'Enable','on');             set(h,'Visible','on');
        h = findobj('Tag','menu_geological_data');
        set(h,'Enable','off');            set(h,'Visible','off');
        %b. open the file to save data
        fid = fopen([PathName,FileName],'w');
        fprintf(fid,'*************************************************\n');
        fprintf(fid,'*\n'); fprintf(fid,'* This file is used to contain the well data. \n');
        fprintf(fid,'*\n'); fprintf(fid,'* Date: ');fprintf(fid,'%s\n',date); fprintf(fid,'*\n');
        fprintf(fid,'*************************************************\n\n');
        status = fclose(fid);
        %c. enable and visible the control menu
        h = findobj('Tag','menu_collection');set(h,'Enable','on');
        h = findobj('Tag','menu_welldata'); set(h,'Enable','on');
        h = findobj('Tag','menu_tools');     set(h,'Enable','on');
        h = findobj('Tag','menu_save_project'); set(h,'Enable','on');
        h = findobj('Tag','menu_save_as_project'); set(h,'Enable','on');
end
```

```
guidata(hObject, handles);

% --------------------------------------------------------------------
function menu_load_project_Callback(hObject, eventdata, handles)
%%%set the state of axes window 'off'
set(handles.axes_well_plot,'Visible','off');
set(get(handles.axes_well_plot,'Children'),'Visible','off');
%%%set the state of left_control window 'off'
h = findobj('Tag','uipanel_left_control_well'); set(h,'Visible','off');
hchildren = get(h,'Children');
for(i=1:1:length(hchildren))        set(hchildren(i),'Visible','off'); end
%%%set the state of bottom_panel window 'off'
h = findobj('Tag','uipanel_bottom_state_well');    set(h,'Visible','off');
hchildren = get(h,'Children');
for(i=1:1:length(hchildren))        set(hchildren(i),'Visible','off'); end
%%%1.set the state of axes window 'off'
set(handles.axes_well_plot,'Visible','off');
set(get(handles.axes_well_plot,'Children'),'Visible','off');
set(handles.axes2_well_plot,'Visible','off');
set(get(handles.axes2_well_plot,'Children'),'Visible','off');
h = findobj('Tag','menu_collection'); set(h,'Enable','off');
h = findobj('Tag','menu_tools'); set(h,'Enable','off'); set(h,'Visible','off');
h = findobj('Tag','menu_save_project');         set(h,'Visible','off');
h = findobj('Tag','menu_save_as_project'); set(h,'Visible','off');
h = findobj('Tag','menu_save_as_project'); set(h,'Visible','off');
h = findobj('Tag','menu_analysis_data'); set(h,'Visible','on');
%%0. dialog for loading well data
[FileName,PathName,FilterIndex]    =    uigetfile({'*.well','Well    file(*.well)';'*.*','All
file(*.*)'});
handles.FileName = FileName;    handles.PathName = PathName;
handles.FilterIndex = FilterIndex;
if isequal(FileName,0) | isequal(PathName,0)
    disp('User selected Cancel');
else
```

```
    disp(['User selected',fullfile(PathName,FileName)]);
            fid = fopen([handles.PathName,handles.FileName],'r');
            for(i = 1:1:9)      tline = fgetl(fid);             end
            wellnumber = 0;             wellnumber = fscanf(fid,'%d');
            for(i=1:1:wellnumber)
                handles.CurrentWell = loadwell(handles.CurrentWell,fid);
                handles.WellArray{i} = handles.CurrentWell;
                if(i ~= wellnumber)
                    tline = fgetl(fid);    tline = fgetl(fid);
                end
            end
            status = fclose(fid);
            h = findobj('Tag','menu_collection');      set(h,'Enable','on');
            h=findobj('Tag','menu_welldata');set(h,'Enable','on');        set(h,'Visible','on');
            h = findobj('Tag','menu_tools');set(h,'Enable','on');      set(h,'Visible','on');
            h = findobj('Tag','menu_save_project');        set(h,'Enable','on');
            h = findobj('Tag','menu_save_as_project');    set(h,'Enable','on');
            h = findobj('Tag','menu_analysis_data');      set(h,'Enable','on');
end
guidata(hObject, handles);

% --------------------------------------------------------------------
function menu_save_project_Callback(hObject, eventdata, handles)
savedata(hObject, eventdata, handles);
% --------------------------------------------------------------------
function menu_save_as_project_Callback(hObject, eventdata, handles)
%0.save currentwell to the wellarray
string_wellname = get(handles.CurrentWell,'WellName');
numberwell = length(handles.WellArray);
    if(numberwell == 0) handles.WellArray{1} = handles.CurrentWell;
    else
        for(i=1:1:numberwell)
            if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
            handles.WellArray{i} = handles.CurrentWell;        break;
```

```
            else
              if(i==numberwell)    handles.WellArray{i+1} = handles.CurrentWell; end
              end
          end       end
guidata(hObject, handles);
          [handles.FileName,handles.PathName,handles.FilterIndex]
              … = uiputfile('*.well','Save Workspace As','Well file(*.well)');
          if isequal(handles.FileName,0) | isequal(handles.PathName,0)
              msgbox('User selected Cancel!','Save As Warning!!','warn');
          else
              fid = fopen([handles.PathName,handles.FileName],'w');
              fprintf(fid,'*******************************************\n');
              fprintf(fid,'*\n');fprintf(fid,'*This file is used to contain well data.\n');
              fprintf(fid,'*\n'); fprintf(fid,'* Date: ');      fprintf(fid,'%s\n',date);
              fprintf(fid,'*\n');
              fprintf(fid,'*******************************************\n\n');
              fprintf(fid,'Total Number of Well\n '); printf(fid,'%d\n\n',numberwell);
              status = fclose(fid);
          end

% --------------------------------------------------------------------
function menu_exit_Callback(hObject, eventdata, handles)
% Call modaldlg with the argument 'Position'.
user_response = exitmodaldlg ('Title','Confirm Close');
switch user_response
case {'No'}
     delete(handles.figure_data_evaluation);          clear all;
case 'Yes'
     if(length(handles.FileName)>0 & length(handles.PathName)>0)
          savedata(hObject, eventdata, handles);
     end
     delete(handles.figure_data_evaluation);
end
% --------------------------------------------------------------------
```

```
function menu_welldata_Callback(hObject, eventdata, handles)
m = InitialWelldataevaluationInterface(handles);


% --------------------------------------------------------------------
% FOR COLLECTIING WELL DATA
function popupmenu_welllist_Callback(hObject, eventdata, handles)
%get the current well name
h = findobj('Tag','popupmenu_welllist'); allwell = get(h,'String');
currentwellname = allwell(get(h,'Value'),:);
currentwellname = deblank(currentwellname);
if(~logical(strcmp(currentwellname,'None')))
      for(i = 1:1:length(handles.WellArray))
            if(logical(strcmp(currentwellname,get(handles.WellArray{i},'WellName'))))
                  handles.CurrentWell = handles.WellArray{i}; break;
            end
      end
end
%initial interface
InitialWelldataevaluationInterface(handles);
h = findobj('Tag','popupmenu_well_dynamic_data');
datatype = set(h,'Value',1); set(handles.pushbutton_load_dynamic_data,'String','Load
Rate');
if(length(handles.CurrentWell.WellPDGPressure) == 0)
    set(handles.pushbutton_view_dynamic_data,'Enable','off');
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function popupmenu_welllist_CreateFcn(hObject, eventdata, handles)
if ispc          set(hObject,'BackgroundColor','white');
else          set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --------------------------------------------------------------------
% --- Executes on button press in pushbutton_add_well.
function pushbutton_add_well_Callback(hObject, eventdata, handles)
```

```
currentwell = wellclass();    handles.CurrentWell = currentwell;
guidata(hObject, handles); InitialWelldataevaluationInterface(handles);
h = findobj('Tag','uipanel_left_control_well'); hchildren = get(h,'Children');
for(i=1:1:length(hchildren))
    set(hchildren(i),'Visible','on');      set(hchildren(i),'Enable','on');
end
h = findobj('Tag','pushbutton_view_dynamic_data');      set(h,'Enable','off');
h = findobj('Tag','frame_well_performance');                set(h,'Enable','off');
h = findobj('Tag','popupmenu_well_dynamic_data');      set(h,'Enable','off');
h = findobj('Tag','pushbutton_load_dynamic_data');        set(h,'Enable','off');
h = findobj('Tag','text_well_dynamic_data');                 set(h,'Enable','off');
WellNumber = length(handles.WellArray);
string_currentwellname = get(currentwell,'WellName');
stringlist = string_currentwellname;
if(WellNumber ~= 0)
        sep = '|';    stringlist = [string_currentwellname];
        for i=1:1:WellNumber-1
            string_wellname = get(handles.WellArray{i},'WellName');
            if(~logical(strcmp(string_wellname,'None')))
                stringlist = [stringlist,sep,string_wellname];
        end     end
        string_wellname = get(handles.WellArray{WellNumber},'WellName');
        if(~logical(strcmp(string_wellname,'None')))
            stringlist = [stringlist,sep,string_wellname];
 end    end
set(findobj('Tag','popupmenu_welllist'),'String',stringlist);
set(findobj('Tag','popupmenu_welllist'),'Value',1);

% --------------------------------------------------------------------
% --- Executes on button press in pushbutton_del_well.
function pushbutton_del_well_Callback(hObject, eventdata, handles)
newwellarray = {}; count = 1;
string_currentwellname = get(handles.CurrentWell,'WellName');
if(logical(strcmp(string_currentwellname,'None')))
```

```
    msgbox('there is no well to be deleted.');
else
    WellNumber = length(handles.WellArray);
    if(WellNumber == 0)
    currentwell = wellclass();          handles.CurrentWell = currentwell;
    else
        for i=1:1:WellNumber
            string_wellname = get(handles.WellArray{i},'WellName');
            if(~logical(strcmp(string_wellname,string_currentwellname)))
                if(i == WellNumber)
                currentwell = wellclass(); handles.CurrentWell = currentwell;
                else
                newwellarray{count} = handles.WellArray{i};count = count + 1;
        end end end
        handles.WellArray = newwellarray;
        if(length(handles.WellArray) == 0)
            currentwell = wellclass(); handles.CurrentWell = currentwell;
        else
            handles.CurrentWell = handles.WellArray{1};
end     end     end
InitialWelldataevaluationInterface(handles);
guidata(hObject, handles);

% --- Executes on button press in pushbutton_load_dynamic_data.
function pushbutton_load_dynamic_data_Callback(hObject, eventdata, handles)
string_wellname = get(findobj('Tag','edit_well_name'),'String');
if(~logical(strcmp(string_wellname,'')) && logical(strcmp(string_wellname,'None')))
    msgbox('Please Input Well Name!!');
else
    dynamicdatatype = get(findobj('Tag','popupmenu_well_dynamic_data'),'Value');
    switch dynamicdatatype
            case 1
    [FileName,PathName,FilterIndex] = uigetfile({'*.cp','PDG Pressure file(*.cp)'});
            if(length(PathName) > 1)
```

```
            temppdgpressure = load([PathName,FileName]);
            handles.CurrentWell = …
            set(handles.CurrentWell,'WellPDGPressure',temppdgpressure);
            WellNumber = length(handles.WellArray);
        for i=1:1:WellNumber
        if(logical(strcmp(string_wellname,get(handles.WellArray{i}…
                ,'WellName'))))
                handles.WellArray{i} = handles.CurrentWell;
            end
        end
            clear temppdgpressure;
            set(findobj('Tag','pushbutton_view_dynamic_data'),'Enable','on');
        else
            msgbox('Please select PDG pressure file name!!!','modal');
        end
  case 2
[FileName,PathName,FilterIndex] = uigetfile({'*.rr','Realtime Rate file(*.rr)'});
  case 3
[FileName,PathName,FilterIndex] = uigetfile({'*.tr','Tubeline Rate file(*.tr)'});
        if(length(PathName) > 1)
            temptubelinedata = load([PathName,FileName]);
            tempWellProduction = handles.CurrentWell.WellProduction;
            tempWellProduction{2} = temptubelinedata;
            handles.CurrentWell = …
            set(handles.CurrentWell,'WellProduction',tempWellProduction);
            WellNumber = length(handles.WellArray);
        for i=1:1:WellNumber
            if(logical(strcmp(string_wellname,…
            get(handles.WellArray{i},'WellName'))))
                handles.WellArray{i} = handles.CurrentWell;
            end
        end
            clear temptubelinedata;    clear tempWellProduction;
            set(findobj('Tag','pushbutton_view_dynamic_data'),'Enable','on');
```

```
            else
            msgbox('Please select tubine-line production rate file name!!!','modal');
        end
    case 4
        [FileName,PathName,FilterIndex] = uigetfile({'*.dr','Cumulated …
         Daily Rate file(*.dr)'});
            if(length(PathName) > 1)
                tempcummulateddata = load([PathName,FileName]);
                tempWellProduction = handles.CurrentWell.WellProduction;
                tempWellProduction{3} = tempcummulateddata;
                handles.CurrentWell =
                        …set(handles.CurrentWell,'WellProduction',tempWellProduction);
            WellNumber = length(handles.WellArray);
            for i=1:1:WellNumber
      if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
                        handles.WellArray{i} = handles.CurrentWell;
                end
            end
                clear tempcummulateddata; clear tempWellProduction;
                set(findobj('Tag','pushbutton_view_dynamic_data'),'Enable','on');
            else
            msgbox('Please select cumulated production rate file name!!!','modal');
            end
     case 5
            PDGpressure = handles.CurrentWell.WellPDGPressure;
            if(length(PDGpressure) > 0)
                    RecoParams = RecoverParams;
                    if(isempty(RecoParams))
                            msgbox('Please input recover parameter!!!','modal');
                    else
                    account = PDGpressure(:,1);      time = PDGpressure(:,2);
                    pressure = PDGpressure(:,3);
                    [recovered_rate,recovered_rate2] = …
                                    recover_function(account,time,pressure,RecoParams);
```

```
                    tempWellProduction = handles.CurrentWell.WellProduction;
                    tempWellProduction{4} = recovered_rate;
                    handles.CurrentWell = …
                set(handles.CurrentWell,'WellProduction',tempWellProduction);
                    WellNumber = length(handles.WellArray);
                    for i=1:1:WellNumber
          if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
                        handles.WellArray{i} = handles.CurrentWell;
                    end    end
                    clear tempWellProduction;
                    set(findobj('Tag','pushbutton_view_dynamic_data'),'Enable','on');
                  end
                else msgbox('Please input the PDG pressure first!!!','modal');    end
    otherwise    disp('Unknow Item');          end
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_view_dynamic_data_Callback(hObject, eventdata, handles)
handles.flag_plotoperation = 1;
handles.flag_graphs = 0;
plotdata_function(hObject, eventdata, handles);
% --------------------------------------------------------------------
function pushbutton_save_well_data_Callback(hObject, eventdata, handles)
string_wellname = get(findobj('Tag','edit_well_name'),'String');
if( ~logical(strcmp(string_wellname,'')) && logical(strcmp(string_wellname,'None')) )
    msgbox('Please Input Well Name!!','Warning');
else
    handles.CurrentWell = set(handles.CurrentWell,'WellName',string_wellname);
    welltypelist = get(findobj('Tag','popupmenu_well_type'),'String');
    strwelltype = welltypelist{get(findobj('Tag','popupmenu_well_type'),'Value')};
    handles.CurrentWell = set(handles.CurrentWell,'WellType',strwelltype);
    wellradius = str2double(get(findobj('Tag','edit_well_radius'),'String'));
    handles.CurrentWell = set(handles.CurrentWell,'WellRadius',wellradius);
formationthickness = str2double(get(findobj('Tag','edit_formation_thickness'),'String'));
```

```
handles.CurrentWell = set(handles.CurrentWell,'WellPerforated',formationthickness);
    porosity = str2double(get(findobj('Tag','edit_formation_porosity'),'String'));
    handles.CurrentWell = set(handles.CurrentWell,'FormationPorosity',porosity);
    numberwell = length(handles.WellArray);
    if(numberwell == 0)
        handles.WellArray{1} = handles.CurrentWell;
    else
        for(i=1:1:numberwell)
        if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
                handles.WellArray{i} = handles.CurrentWell; break;
            else
                if(i==numberwell)
                    handles.WellArray{i+1} = handles.CurrentWell;
                end end end
    end
    InitialWelldataevaluationInterface(handles);
    h = findobj('Tag','popupmenu_well_dynamic_data');       set(h,'Value',1);
    set(handles.pushbutton_load_dynamic_data,'String','Load Data');
    guidata(hObject, handles);
end
% --------------------------------------------------------------------
function popupmenu_data_type_Callback(hObject, eventdata, handles)
if(get(handles.popupmenu_data_type,'Value') == 1 …
&& get(handles.popupmenu_evaluation,'Value') == 1)
   set(handles.popupmenu_multi_plot,'Enable','on');
   set(handles.text_multi_plot,'Enable','on');
else
   set(handles.popupmenu_multi_plot,'Enable','off');
   set(handles.text_multi_plot,'Enable','off');
end
% --------------------------------------------------------------------
function popupmenu_evaluation_Callback(hObject, eventdata, handles)
if(get(handles.popupmenu_data_type,'Value') == 1…
&& get(handles.popupmenu_evaluation,'Value') == 1)
```

```
    set(handles.popupmenu_multi_plot,'Enable','on');
    set(handles.text_multi_plot,'Enable','on');
else
    set(handles.popupmenu_multi_plot,'Enable','off');
    set(handles.text_multi_plot,'Enable','off');
end
% --------------------------------------------------------------------
function pushbutton_plot_dynamcidata_Callback(hObject, eventdata, handles)
handles.flag_plotoperation = 2;
handles.flag_graphs = 0;
plotdata_function(hObject, eventdata, handles);
% --------------------------------------------------------------------
function menu_fluid_rock_data_Callback(hObject, eventdata, handles)
set(handles.axes_well_plot,'Visible','off');
set(get(handles.axes_well_plot,'Children'),'Visible','off');
set(handles.axes2_well_plot,'Visible','off');
set(get(handles.axes2_well_plot,'Children'),'Visible','off');
set(handles.uipanel_left_control_well,'Visible','off');
set(handles.uipanel_bottom_state_well,'Visible','off');
wellfluid = FluidAndRock(handles.CurrentWell);
handles.CurrentWell = wellfluid;      guidata(hObject, handles);
```

Data evaluation function

```
function varargout = DataEvaluation_function(DataEval,hObject,handles);
%1.select the data type
%         'original data' ----- evaluate the pressure or rate;
%         'data interval' ----- evaluate the quality of data sampling;
%2.select the evaluated method
%         'normal'        ----- normal plot;*fault*
%         'frequency' ----- show the frequency of every data range;
%         'PDF'             ----- probability density function;
%         'CDF'             ----- cumulative distribution function;
%3.evaluation_range      'xmin' and 'xmax'
set(handles.axes_well_plot,'Visible','off');
set(get(handles.axes_well_plot,'Children'),'Visible','off');
```

```
set(handles.axes2_well_plot,'Visible','off');
set(get(handles.axes2_well_plot,'Children'),'Visible','off');
c = logical(strcmp(get(handles.popupmenu_multi_plot,'Visible'),'on'));
b = logical((get(handles.popupmenu_multi_plot,'Value')…
~= get(handles.popupmenu_well_dynamic_data,'Value')));
if(b && c)
    count = 1;      xmin=DataEval{3};       xmax=DataEval{4};
    if( get(handles.popupmenu_multi_plot,'Value') >
     get(handles.popupmenu_well_dynamic_data,'Value') )
        dataselect =[get(handles.popupmenu_multi_plot,'Value'),
       get(handles.popupmenu_well_dynamic_data,'Value')];
    else
        dataselect = [get(handles.popupmenu_well_dynamic_data,'Value'),
       …get(handles.popupmenu_multi_plot,'Value')];
    end
    plottemp = {[];[]};       temp = [];      stringtitle = '';
    for(i = 1:1:length(plottemp))
    dynamicdataselect = dataselect(i);
    switch dynamicdataselect
    case 1
        A = handles.CurrentWell.WellPDGPressure;
        if(length(A) > 1)
            jmin = find(A(:,2)>=xmin,1); jmax = find(A(:,2)>=xmax,1);
            if(isempty(jmax))     jmax = length(A(:,2));   end
            temp(1:jmax-jmin+1,1) =   A(jmin:jmax,1);
            temp(1:jmax-jmin+1,2) =   A(jmin:jmax,2);
            temp(1:jmax-jmin+1,3) =   A(jmin:jmax,3);
            clear A;
        else
            msgbox('Thare is no PDG pressure data.');       break;
        end
    case 2          break;
    case 3
        A = handles.CurrentWell.WellProduction;          A = A{2};
```

```
        if(length(A) > 1)
            jmin = find(A(:,2)>=xmin,1);        jmax =find(A(:,2)>=xmax,1);
            if(isempty(jmax))        jmax = length(A(:,2));        end
            temp(1:jmax-jmin+1,1) =    A(jmin:jmax,1);
            temp(1:jmax-jmin+1,2) =    A(jmin:jmax,2);
            temp(1:jmax-jmin+1,3) =    A(jmin:jmax,3);
            clear A;
        else
            msgbox('Thare is no tubeline production DATA.');        break;
        end
    case 4
        A = handles.CurrentWell.WellProduction;                A = A{3};
        if(length(A) > 1)
            jmin = find(A(:,1)>=xmin,1);    jmax = find(A(:,1)>=xmax,1);
            if(isempty(jmax))          jmax = length(A(:,1));       end
            temp(1:jmax-jmin+1,1) =    A(jmin:jmax,1);
            temp(1:jmax-jmin+1,2) =    A(jmin:jmax,2);
            temp(1:jmax-jmin+1,3) =    A(jmin:jmax,3);                    clear A;
        else
            msgbox('Thare is no daily rate DATA.');                break;
        end
    case 5
        A = handles.CurrentWell.WellProduction;                A = A{4};
        if(length(A) > 1)
            jmin = find(A(:,2)>=xmin,1); jmax = find(A(:,2)>=xmax,1);
            if(isempty(jmax)) jmax = length(A(:,2)); end
            temp(1:jmax-jmin+1,1) =    A(jmin:jmax,1);
            temp(1:jmax-jmin+1,2) =    A(jmin:jmax,2);
            temp(1:jmax-jmin+1,3) =    A(jmin:jmax,3);          clear A;
        else
            msgbox('Thare is no recovered rate DATA.');       break;
        end                otherwise
    end
    plottemp{i} = temp;         temp = [];
```

```
   end
if(handles.flag_graphs == 1)
      figure;        axes;        ax1 = gca;        set(ax1,'XColor','r','YColor','r');
      ax2 = axes('Position',get(ax1,'Position'),'XAxisLocation','top',...
                'YAxisLocation','right','Color','none','XColor','k','YColor','k');
      axesarry = [ax1,ax2];
end
if(handles.flag_graphs == 0)
      axesarry = [handles.axes_well_plot,handles.axes2_well_plot];
end
for(i=1:1:2)
      temp = plottemp{i};
      switch dataselect(i)
      case 1
            if(length(temp) > 1)
               set(axesarry(i),'Visible','on');       axes(axesarry(i));
               if(i == 1)
                     line(temp(:,2),temp(:,3),'Color','r');    ax1 = gca;
                     set(ax1,'XColor','r','YColor','r'); stringtitle = [' PDG Pressure '];
               else
                     ax2 = gca;    hl2 = line(temp(:,2),temp(:,3),'Color','k','Parent',ax2);
                     set(get(axesarry(i),'Ylabel'),'String','Pressure(psi)',…
          'FontWeight','bold','FontSize',12); stringtitle = [stringtitle,' PDG Pressure '];
               end          end
      case 2               break;
      case 3
             if(length(temp) > 1)
                set(axesarry(i),'Visible','on');            axes(axesarry(i));
            if(i == 1)
                     line(temp(:,2),temp(:,3),'Color','r');
                     set(get(axesarry(i),'Xlabel'),'String','Time(hours)',…
                     'FontWeight','bold','FontSize',12);
                     set(get(axesarry(i),'Ylabel'),'String','Production …
                     Rate(mmscf/day)','FontWeight','bold','FontSize',12);
```

```
                    ax1 = gca;              set(ax1,'XColor','r','YColor','r');
                    stringtitle = [' Tubeline Rate '];
                else
                    ax2 = gca; hl2 = line(temp(:,2),temp(:,3),'Color','k','Parent',ax2);
                    set(get(axesarry(i),'Ylabel'),'String','Production …
                    Rate(mmscf/day)','FontWeight','bold','FontSize',12);
                    stringtitle = [stringtitle,' Tubeline Rate '];
                end             end
    case 4
            if(length(temp) > 1)
                set(axesarry(i),'Visible','on');    axes(axesarry(i));
                if(i == 1)
                    stairs(temp(:,1),temp(:,2),'r');
                    set(get(axesarry(i),'Xlabel'),'String','Time(hours)'…
                    ,'FontWeight','bold','FontSize',12);
                    set(get(axesarry(i),'Ylabel'),'String','Production
                    Rate(mmscf/day)','FontWeight','bold','FontSize',12);
                    ax1 = gca; set(ax1,'XColor','r','YColor','r');
                    stringtitle = [' Daily Gas Rate '];
                else
                    ax2 = gca; hl2 = stairs(temp(:,1),temp(:,2),'Color','k','Parent',ax2);
                    set(get(axesarry(i),'Ylabel'),'String','Production
                    Rate(mmscf/day)','FontWeight','bold','FontSize',12);
                    stringtitle = [stringtitle,' Daily Gas Rate '];
                end       end
     case 5
            if(length(temp) > 1)
                set(axesarry(i),'Visible','on');      axes(axesarry(i));
                if(i == 1)
                stairs(temp(:,1),temp(:,2),'r');
set(get(axesarry(i),'Xlabel'),'String','Time(hours)','FontWeight','bold','FontSize',12);
                    set(get(axesarry(i),'Ylabel'),'String','Production
                    Rate(mmscf/day)','FontWeight','bold','FontSize',12);
                    ax1 = gca; set(ax1,'XColor','r','YColor','r');
```

```
                    stringtitle = [' Recovered Gas Rate '];
                else
                    ax2 = gca; hl2 = line(temp(:,1),temp(:,2),'Color','k','Parent',ax2);
                    set(get(axesarry(i),'Ylabel'),'String','Production
                    Rate(mmscf/day)','FontWeight','bold','FontSize',12);
                    stringtitle = [stringtitle,' Recovered Gas Rate '];
                end;    end
    otherwise
    end
    if(i == 1)              stringtitle = [stringtitle,' VS ']; end
    title('','FontWeight','bold','FontSize',14);    box off;
end
title(stringtitle,'FontWeight','bold','FontSize',14);
else
set(handles.axes_well_plot,'Visible','off');
set(get(handles.axes_well_plot,'Children'),'Visible','off');
set(handles.axes2_well_plot,'Visible','off');
set(get(handles.axes2_well_plot,'Children'),'Visible','off');
count = 1; xmin=DataEval{3}; xmax=DataEval{4};
set(handles.axes2_well_plot,'Visible','off');
set(get(handles.axes2_well_plot,'Children'),'Visible','off');
dynamicdataselect = get(handles.popupmenu_well_dynamic_data,'Value')
switch dynamicdataselect
  case 1
    A = handles.CurrentWell.WellPDGPressure;
    numberA = A(:,1);     timeA = A(:,2);
    pressureA = A(:,3);   totalsize =length(timeA);
    jmin = find(timeA>=xmin,1);     jmax = find(timeA>=xmax,1);
    if(isempty(jmax))
        jmax = length(A(:,2));
    end
    plotdata(1:jmax-jmin+1,1) =  numberA(jmin:jmax);
    plotdata(1:jmax-jmin+1,2) =  timeA(jmin:jmax);
    plotdata(1:jmax-jmin+1,3) =  pressureA(jmin:jmax);     clear A;
```

```
    case 3
        A = handles.CurrentWell.WellProduction;
        A = A{2};          numberA = A(:,1);
        timeA = A(:,2);        rateA = A(:,3);
        totalsize =length(timeA);
        jmin = find(timeA>=xmin,1);        jmax = find(timeA>=xmax,1);
        if(isempty(jmax))                  jmax = length(A(:,2));          end
        plotdata(1:jmax-jmin+1,1) =    numberA(jmin:jmax);
        plotdata(1:jmax-jmin+1,2) =    timeA(jmin:jmax);
        plotdata(1:jmax-jmin+1,3) =    rateA(jmin:jmax);          clear A;
    case 4
        % daily rate
        A = handles.CurrentWell.WellProduction;        A = A{3};
        timedialy = A(:,1);        gasrate = A(:,2);        oilrate = A(:,3);
        totalsize =length(timedialy);
        jmin = find(timedialy>=xmin,1);        jmax = find(timedialy>=xmax,1);
        if(isempty(jmax))                    jmax = length(A(:,2));          end
        plotdata(1:jmax-jmin+1,1) =    oilrate(jmin:jmax);
        plotdata(1:jmax-jmin+1,2) =    timedialy(jmin:jmax);
        plotdata(1:jmax-jmin+1,3) =    gasrate(jmin:jmax);            clear A;
    case 5
        A = handles.CurrentWell.WellProduction;
        A = A{4};        time = A(:,1);        gasrate = A(:,2);        totalsize =length(time);
        jmin = find(time>=xmin,1);        jmax = find(time>=xmax,1);
        if(isempty(jmax))                    jmax = length(A(:,2));        end
        plotdata(1:jmax-jmin+1,1) =    time(jmin:jmax);
        plotdata(1:jmax-jmin+1,2) =    gasrate(jmin:jmax);          clear A;
        otherwise
end
        if(logical(strcmp(DataEval{1},'Original Data')))
            method = DataEval{2}
            switch method
                case 1
                    if(handles.flag_graphs == 1)        figure;              end
```

```
        if(handles.flag_graphs == 0)
            axes(handles.axes_well_plot);
            clear handles.plotline;
        end
        switch dynamicdataselect
        case 1
            plot(plotdata(:,2),plotdata(:,3),'b*-');
            %varargout{1} = handles.plotline;
            xlabel('Time(hours)','FontWeight','bold','FontSize',12);
            ylabel('Pressure(psi)','FontWeight','bold','FontSize',12);
           title('PDG Pressure vs Time','FontWeight','bold','FontSize',14);
        case 3
            plot(plotdata(:,2),plotdata(:,3),'b*-');
            xlabel('Time(hours)','FontWeight','bold','FontSize',12);
ylabel('Production Rate(mmscf/day)','FontWeight','bold','FontSize',12);
title('Tubeline Rate vs Time','FontWeight','bold','FontSize',14);
        case 4
            stairs(plotdata(:,2),plotdata(:,3),'r*-');
            xlabel('Time(hours)','FontSize',14,'color','k');
            ylabel('Gas Rate(mmscf/d)','FontSize',14,'color','k');
            title('Daily Gas Rate','FontSize',14);
        case 5
           stairs(plotdata(:,1),plotdata(:,2),'r*-');
            xlabel('Time(hours)','FontSize',14,'color','k');
            ylabel('Gas Rate(mmscf/d)','FontSize',14,'color','k');
            title('Recovered Gas Rate','FontSize',14); otherwise
        end
case 2
        if(handles.flag_graphs == 1)    figure;    end
        if(handles.flag_graphs == 0)
            axes(handles.axes_well_plot);      clear handles.plotline;
        end
        hist(plotdata(:,3),1000);
        title('Distribution of data values','FontSize',14);
```

```
        ylabel('Counts','FontWeight','bold','FontSize',12);
        switch dynamicdataselect
        case 1
            xlabel('Pressure(psi)','FontWeight','bold','FontSize',12);
            title('PDG Pressure vs Time','FontWeight','bold','FontSize',14);
        case 3
      xlabel('Tubeline Rate(mmscf/day)','FontWeight','bold','FontSize',12);
            title('Tubeline Rate vs Time','FontWeight','bold','FontSize',14);
        case 4
    xlabel('Daily Gas Rate(mmscf/day)','FontWeight','bold','FontSize',12);
           title('Daily Gas Rate vs Time','FontWeight','bold','FontSize',14);
        case 5
  xlabel('Recovered Rate(mmscf/day)','FontWeight','bold','FontSize',12);
     title('Recovered Gas Rate vs Time','FontWeight','bold','FontSize',14);
              otherwise                          end
  case 3
        if(handles.flag_graphs == 1)    figure;    end
        if(handles.flag_graphs == 0)
              axes(handles.axes_well_plot);
              clear handles.plotline;
        end
        normplot(plotdata(:,3));
        title('Distribution of data values','FontSize',14);
        ylabel('Counts','FontWeight','bold','FontSize',12);
        switch dynamicdataselect
        case 1
              xlabel('Pressure(psi)','FontWeight','bold','FontSize',12);
            title('PDG Pressure vs Time','FontWeight','bold','FontSize',14);
        case 3
      xlabel('Tubeline Rate(mmscf/day)','FontWeight','bold','FontSize',12);
            title('Tubeline Rate vs Time','FontWeight','bold','FontSize',14);
        case 4
    xlabel('Daily Gas Rate(mmscf/day)','FontWeight','bold','FontSize',12);
    title('Daily Gas Rate vs Time','FontWeight','bold','FontSize',14);
```

```
                    case 5
            xlabel('Recovered Gas Rate(mmscf/day)','FontWeight','bold','FontSize',12);
            title('Recovered Gas Rate vs Time','FontWeight','bold','FontSize',14);
                        otherwise                           end
                otherwise               end
        else
            plotdata(1,3) = 0;              totalsize = length(plotdata);
            for    i = 2:1:totalsize     plotdata(i,3) = plotdata(i,2)-plotdata(i-1,2);     end
            method = DataEval{2};
            switch method
                case 1
                    if(handles.flag_graphs == 1)        figure;         end
                    if(handles.flag_graphs == 0)
                        axes(handles.axes_well_plot); clear handles.plotline;
                    end
                    plot(plotdata(:,2),plotdata(:,3),'b*-');
                    ylabel('Time Interval(hours)','FontWeight','bold','FontSize',12);
                    xlabel('Time(hours)','FontWeight','bold','FontSize',12);
                    title('Data Interval vs Time','FontWeight','bold','FontSize',14);
                case 2
                    if(handles.flag_graphs == 1)       figure;    end
                    if(handles.flag_graphs == 0)
                        axes(handles.axes_well_plot);       clear handles.plotline;
                    end
                case 3
                    if(handles.flag_graphs == 1)     figure;            end
                    if(handles.flag_graphs == 0)
                        axes(handles.axes_well_plot);    clear handles.plotline;
                    end
                  normplot(plotdata(:,3));title('Distribution of data values','FontSize',14);
                    ylabel('Counts','FontWeight','bold','FontSize',12);
                    xlabel('Time Interval(hours)','FontWeight','bold','FontSize',12);
                otherwise               end          end
end
```

```
guidata(hObject, handles);      return;
```

Initial data collection Interface function

```
%%%
```

```
            string_wellname = get(handles.WellArray{WellNumber},'WellName');
            stringlist = [stringlist,string_wellname];
        end
        set(findobj('Tag','popupmenu_welllist'),'String',stringlist);
        set(findobj('Tag','popupmenu_welllist'),'Value',1);
        h = findobj('Tag','edit_well_name');                set(h,'String','None');
        h = findobj('Tag','popupmenu_well_type');       set(h,'Value',1);
        h = findobj('Tag','edit_well_radius');              set(h,'String','None');
        h = findobj('Tag','edit_formation_thickness');  set(h,'String','None');
        h = findobj('Tag','edit_formation_porosity');     set(h,'String','None');
    else
        set(handles.axes_well_plot,'Visible','off');
        set(get(handles.axes_well_plot,'Children'),'Visible','off');
        set(handles.axes2_well_plot,'Visible','off');
        set(get(handles.axes2_well_plot,'Children'),'Visible','off');
        h = findobj('Tag','uipanel_bottom_state_well');             set(h,'Visible','off');
        hchildren = get(h,'Children');
        for(i=1:1:length(hchildren))    set(hchildren(i),'Visible','off');  end
        h = findobj('Tag','uipanel_left_control_well');       set(h,'Visible','on');
        hchildren = get(h,'Children');
        for(i=1:1:length(hchildren))
            set(hchildren(i),'Visible','on');    set(hchildren(i),'Enable','on');
        end
        WellNumber = length(handles.WellArray);
        string_currentwellname = get(currentwell,'WellName');
        stringlist = string_currentwellname;
        if(WellNumber ~= 0)
            sep = '|';  stringlist = [string_currentwellname];
            for i=1:1:WellNumber
                string_wellname = get(handles.WellArray{i},'WellName');
                if(~logical(strcmp(string_wellname,string_currentwellname)))
                    stringlist = [stringlist,sep,string_wellname];
                end             end
        end
```

```
            set(findobj('Tag','popupmenu_welllist'),'String',stringlist);
            set(findobj('Tag','popupmenu_welllist'),'Value',1);
            h = findobj('Tag','edit_well_name');
            set(h,'String',get(currentwell,'WellName'));
            h = findobj('Tag','popupmenu_well_type');
            welltype = get(currentwell,'WellType');
            switch welltype
                case 'OilWell'                  set(h,'Value',1);
                case 'GasWell'                  set(h,'Value',2);
                case 'WaterWell'                set(h,'Value',3);
                case 'MixtureWell'              set(h,'Value',4);
                otherwise                       disp('');
            end
            h = findobj('Tag','edit_well_radius');
            set(h,'String',num2str(get(currentwell,'WellRadius')));
             h = findobj('Tag','edit_formation_thickness');
            set(h,'String',num2str(get(currentwell,'WellPerforated')));
            h = findobj('Tag','edit_formation_porosity');
            set(h,'String',num2str(get(currentwell,'FormationPorosity')));
            if(length(get(currentwell,'WellPDGPressure')) == 0)
                h = findobj('Tag','pushbutton_view_dynamic_data'); set(h,'Enable','off');
            else
                h = findobj('Tag','pushbutton_view_dynamic_data'); set(h,'Enable','on');
            end         end
        return;
else
        isclass = 0;        return;
end
```

Save data function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           this function is used to save data to file.
function savedata(hObject, eventdata, handles)
string_wellname = get(handles.CurrentWell,'WellName');
numberwell = length(handles.WellArray);
```

```
    if(numberwell == 0)
        handles.WellArray{1} = handles.CurrentWell;
    else
        for(i=1:1:numberwell)
            if(logical(strcmp(string_wellname,…
                get(handles.WellArray{i},'WellName'))))
                handles.WellArray{i} = handles.CurrentWell; break;
            else
                if(i==numberwell)
                    handles.WellArray{i+1} = handles.CurrentWell;
                end   end   end
    end
guidata(hObject, handles);
fid = fopen([handles.PathName,handles.FileName],'w');
fprintf(fid,'*************************************************************\n');
fprintf(fid,'*\n'); fprintf(fid,'*       This file is used to contain the well data. \n');
fprintf(fid,'*\n'); fprintf(fid,'*   Date: ');     fprintf(fid,'%s\n',date);   fprintf(fid,'*\n');
fprintf(fid,'*************************************************************\n\n');
fprintf(fid,'Total Number of Well \n');   fprintf(fid,'%d\n\n',numberwell);
for(i = 1:1:numberwell)
    fprintf(fid,'************************* the ');
    fprintf(fid,'%d',i);
    fprintf(fid,' well*************************\n');
    fid = savewell(handles.WellArray{i},fid);        fprintf(fid,'\n\n');
end
status = fclose(fid);
```

plot data function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            this function is used to plot graphs in axis or figures;
function varargout = plotdata_function(hObject, eventdata, handles);
flag_plotoperation = handles.flag_plotoperation;
flag_graphs = handles.flag_graphs;
switch flag_plotoperation
```

```
case 1
    set(handles.uipanel_bottom_state_well,'Visible','on');
    hchildren = get(handles.uipanel_bottom_state_well,'Children');
    for(i=1:1:length(hchildren))    set(hchildren(i),'Visible','on');    end
    listdynamic = get(findobj('Tag','popupmenu_well_dynamic_data'),'Value');
    switch listdynamic
    case 1
        temp = handles.CurrentWell.WellPDGPressure;
        lengthdata = length(temp);
        if( lengthdata ~= 0)
        set(findobj('Tag','edit_range_min_data'),'String',num2str(temp(1,2)));
  set(findobj('Tag','edit_range_max_data'),'String',num2str(temp(lengthdata,2)));
        set(findobj('Tag','edit_select_min_data'),'String',num2str(temp(1,2)));
   set(findobj('Tag','edit_select_max_data'),'String',num2str(temp(lengthdata,2)));
        set(handles.popupmenu_data_type,'Value',1);
         set(handles.popupmenu_evaluation,'Value',1);
         set(handles.popupmenu_multi_plot,'Value',…
         get(handles.popupmenu_well_dynamic_data,'Value'));
        DataEval{1} = 'Original Data';    DataEval{2} = 1;
        DataEval{3} = temp(1,2);          DataEval{4} = temp(lengthdata,2);
        DataEvaluation_function(DataEval,hObject,handles);
        End                    clear temp;
    case 3
        temp = handles.CurrentWell.WellProduction;
        temp = temp{2};      lengthdata = length(temp);
        if( lengthdata ~= 0)
          set(findobj('Tag','edit_range_min_data'),'String',num2str(temp(1,2)));
   set(findobj('Tag','edit_range_max_data'),'String',num2str(temp(lengthdata,2)));
          set(findobj('Tag','edit_select_min_data'),'String',num2str(temp(1,2)));
   set(findobj('Tag','edit_select_max_data'),'String',num2str(temp(lengthdata,2)));
            set(handles.popupmenu_data_type,'Value',1);
            set(handles.popupmenu_evaluation,'Value',1);
            set(handles.popupmenu_multi_plot,'Value',…
            get(handles.popupmenu_well_dynamic_data,'Value'));
```

```
                DataEval{1} = 'Original Data';                DataEval{2} = 1;
                DataEval{3} = temp(1,2);     DataEval{4} = temp(lengthdata,2);
                DataEvaluation_function(DataEval,hObject,handles);
            End                    clear temp;
        case 4
            temp = handles.CurrentWell.WellProduction;       temp = temp{3};
            lengthdata = length(temp);
            if( lengthdata ~= 0)
            set(findobj('Tag','edit_range_min_data'),'String',num2str(temp(1,1)));
    set(findobj('Tag','edit_range_max_data'),'String',num2str(temp(lengthdata,1)));
            set(findobj('Tag','edit_select_min_data'),'String',num2str(temp(1,1)));
    set(findobj('Tag','edit_select_max_data'),'String',num2str(temp(lengthdata,1)));
            set(handles.popupmenu_data_type,'Value',1);
            set(handles.popupmenu_evaluation,'Value',1);
            set(handles.popupmenu_multi_plot,'Value',…
            get(handles.popupmenu_well_dynamic_data,'Value'));
            DataEval{1} = 'Original Data';               DataEval{2} = 1;
            DataEval{3} = temp(1,1);        DataEval{4} = temp(lengthdata,1);
            DataEvaluation_function(DataEval,hObject,handles);
            End                    clear temp;
        case 5
            temp = handles.CurrentWell.WellProduction;       temp = temp{4};
            lengthdata = length(temp);
            if( lengthdata ~= 0)
              set(handles.popupmenu_data_type,'Value',1);
              set(handles.popupmenu_evaluation,'Value',1);
              DataEval{1} = 'Original Data';             DataEval{2} = 1;
              DataEval{3} = temp(1,1);        DataEval{4} = temp(lengthdata,1);
              DataEvaluation_function(DataEval,hObject,handles);
            end
            clear temp;             otherwise
end
guidata(hObject, handles);
case 2
```

```
            %%clear all axis and display the axes      %% the condition of selection
            timemin    = str2num(get(handles.edit_range_min_data,'String'));
            timemax      = str2num(get(handles.edit_range_max_data,'String'));
            timebegin    = str2num(get(handles.edit_select_min_data,'String'));
            timeend      = str2num(get(handles.edit_select_max_data,'String'));
            if((timebegin >= timeend) | (timebegin >= timemax) |(timeend <= timemin))
                set(handles.axes_well_plot,'Visible','off');
                set(get(handles.axes_well_plot,'Children'),'Visible','off');
                set(handles.axes2_well_plot,'Visible','off');
                set(get(handles.axes2_well_plot,'Children'),'Visible','off');
                msgbox('Wrong selected time!!');
            else
            %% get data types
            datatypes = get(handles.popupmenu_data_type,'String');
            types = datatypes(get(handles.popupmenu_data_type,'Value'));
            %% get evaluation methods
            evaluatiomethod = get(handles.popupmenu_evaluation,'String');
            method= evaluatiomethod(get(handles.popupmenu_evaluation,'Value'));
            %% plot the data
            DataEval{1} = types;
            DataEval{2} = get(handles.popupmenu_evaluation,'Value');
            DataEval{3} = timebegin;
            DataEval{4} = timeend;
            DataEvaluation_function(DataEval,hObject,handles);
            end
        otherwise
end
return;
```

Data processing code

Main function

```
% ------------------------------------------------------------------
function varargout = data_processing_module(varargin)
% DATA_PROCESSING_MODULE M-file for data_processing_module.fig
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',          mfilename, ...
                   'gui_Singleton',    gui_Singleton, ...
                   'gui_OpeningFcn', @data_processing_module_OpeningFcn, ...
                   'gui_OutputFcn',    @data_processing_module_OutputFcn, ...
                   'gui_LayoutFcn',    [] , ...
                   'gui_Callback',     []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% --------------------------------------------------------------------
function data_processing_module_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
% --------------------------------------------------------------------
function varargout = data_processing_module_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% --------------------------------------------------------------------
function pushbutton1_Callback(hObject, eventdata, handles)
data_processing;
delete(handles.figure_data_processing_module);
% --------------------------------------------------------------------
function pushbutton2_Callback(hObject, eventdata, handles)
data_processing_eps;
delete(handles.figure_data_processing_module);
```

Data processing main function

```
% --------------------------------------------------------------------
function varargout = data_processing_eps(varargin)
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',              mfilename, ...
                   'gui_Singleton',    gui_Singleton, ...
                   'gui_OpeningFcn', @data_processing_eps_OpeningFcn, ...
                   'gui_OutputFcn',    @data_processing_eps_OutputFcn, ...
                   'gui_LayoutFcn',    [] , ...
                   'gui_Callback',      []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% ------------------------------------------------------------------
function data_processing_eps_OpeningFcn(hObject, eventdata, handles, varargin)
handles.currenteventnum = 0;
handles.eventnum = -1;
handles.figurecontrol = 0;
guidata(hObject, handles);
ax1 = handles.axes_data_processing;
ax2 = axes('Position',get(ax1,'Position'),'XAxisLocation','top',...
                   'YAxisLocation','right','Color','none','XColor','k','YColor','k');
set(ax2,'visible','off'); handles.axes_data_processing2 = ax2;
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)          handles = outhandles; end
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
handles.output = hObject;
guidata(hObject, handles);
% ------------------------------------------------------------------
function varargout = data_processing_eps_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% ------------------------------------------------------------------
function menu_load_signal_Callback(hObject, eventdata, handles) %#ok<INUSL>
```

```
handles.eventnum = 0;          guidata(hObject, handles);
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)            handles = outhandles; end
guidata(hObject, handles);
  [state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function menu_save_signal_Callback(hObject, eventdata, handles)
handles.currenteventnum = handles.eventnum;
handles.eventnum = 9;    guidata(hObject, handles);
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)          handles = outhandles;      end
handles.eventnum = handles.currenteventnum;      guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_save_as_Callback(hObject, eventdata, handles)
handles.currenteventnum = handles.eventnum;
handles.eventnum = 8;    guidata(hObject, handles);
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)          handles = outhandles;    end
handles.eventnum = handles.currenteventnum;
guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_exit_Callback(hObject, eventdata, handles)
user_response = exitmodaldlg('Title','Confirm Close');
switch user_response
case {'No'}
      delete(handles.figure_data_processing);          clear all;
case 'Yes'
      if(length(handles.FileName)>0 & length(handles.PathName)>0)
            %savedata(hObject, eventdata, handles);
      end
      delete(handles.figure_data_processing);          clear all;
end
```

```
% --------------------------------------------------------------------
function pushbutton_data_preprocessing_Callback(hObject, eventdata, handles)
handles.eventnum = 1;        handles.figurecontrol = 0;
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)        handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_data_interpolation_Callback(hObject, eventdata, handles)
handles.eventnum = 2;      handles.figurecontrol = 0;
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_data_interpolation_Callback(hObject, eventdata, handles)
handles.eventnum = 2;
  [state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)        handles = outhandles; end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_data_decomposition_Callback(hObject, eventdata, handles)
handles.eventnum = 3; handles.figurecontrol = 0;
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)              handles = outhandles;      end
guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_data_decomposition_Callback(hObject, eventdata, handles)
handles.eventnum = 3; handles.figurecontrol = 0;
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)            handles = outhandles; end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_data_evaluation_Callback(hObject, eventdata, handles)
handles.eventnum = 4; handles.figurecontrol = 0;
[state outhandles]= DataProcessingDataManageeps(handles);
```

```
if(state == 1)          handles = outhandles; end
guidata(hObject, handles);
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles;      end
guidata(hObject, handles);


% --------------------------------------------------------------------
function Menu_data_evaluation_Callback(hObject, eventdata, handles)
handles.eventnum = 4; handles.figurecontrol = 0;
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)          handles = outhandles;    end
guidata(hObject, handles);
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_outlier_removal_Callback(hObject, eventdata, handles)
% current button
handles.eventnum = 5; handles.figurecontrol = 0;
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)            handles = outhandles;      end
guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_outlier_removal_Callback(hObject, eventdata, handles)
handles.eventnum = 5;    handles.figurecontrol = 0;
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)          handles = outhandles; end
guidata(hObject, handles);
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_identify_event_Callback(hObject, eventdata, handles)
handles.currentprocessingarray =    struct('Num',0,'periodtype',1,...
```

```
        'periodnum',0,'Originaldata',[],'reducedata',[],'denoisedata',[]);
handles.furtherprocessingarray = [];
handles.eventnum = 6;
switch handles.FilterIndex
        case 1
                temp = handles.buddevent.parameter;
                if(temp(1) == 0)
                        temp(1) = abs(handles.EvaluationData.Mean) + …
                         abs(handles.EvaluationData.Stddata);
                        handles.buddevent.parameter = temp;
                end
        case 2
                tempparameter = handles.buddevent.parameter;
                temp = handles.OriginalPressure.Preprocessingdata;
                if(temp(length(temp),1)<2)
                        if(tempparameter(1) == 0)
                                tempparameter(1) = abs(handles.EvaluationData.Mean) + …
                                 abs(handles.EvaluationData.Stddata)*0.8;
                                tempparameter(2) = 0.0002;          tempparameter(3) = 1;
                                handles.buddevent.parameter = tempparameter;
                        end          end
end
guidata(hObject, handles); handles.figurecontrol = 0;
  [state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_identify_Callback(hObject, eventdata, handles)
handles.eventnum = 6;      handles.figurecontrol = 0;
  [state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)              handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function pushbutton_recovering_rate_Callback(hObject, eventdata, handles)
```

```
handles.eventnum = 10;    handles.figurecontrol = 0;
  [state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)            handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function Menu_recovery_rate_Callback(hObject, eventdata, handles)
handles.eventnum = 10;    handles.figurecontrol = 0;
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)              handles = outhandles;    end
guidata(hObject, handles);
% --------------------------------------------------------------------
function parameter1_input_Callback(hObject, eventdata, handles)
if handles.eventnum == 4
      handles.figurecontrol = 0;
      intermethod = {'hist';'probability density';'Cumulative distribution'};
      temp = get(handles.parameter1_input,'Value');
      handles.EvaluationData.Type = intermethod{temp};
      guidata(hObject, handles);
      [state outhandles] = UpdateDataProcessingInterfaceeps(handles);
      if(state == 1)              handles = outhandles;       end
      guidata(hObject, handles);
end
if handles.eventnum == 10
      temp = get(handles.parameter1_input,'Value');
      periodtype = 0;
      switch temp
            case 1                        periodtype = 1;
            case 2                        periodtype = -1;
            case 3                        periodtype = -2;
      end
      periodnum = 1;
      tempdata =    struct('Num',0,'periodtype',0,'periodnum',…
      0,'Originaldata',[],'reducedata',[],'denoisedata',[]);
      handles.currentprocessingarray = tempdata;
```

```
for i=1:1:length(handles.furtherprocessingarray)
        tempdata = handles.furtherprocessingarray(i);
        if(tempdata.periodtype == periodtype) && …
          (tempdata.periodnum == periodnum)
                handles.currentprocessingarray = tempdata;              break;
        else
                tempdata =    struct('Num',0,'periodtype',0,'periodnum'…
                ,0,'Originaldata',[],'reducedata',[],'denoisedata',[]);
        end
end
if(handles.currentprocessingarray.Num == 0)
  tempdata.periodtype = periodtype; handles.currentprocessingarray = tempdata;
end
guidata(hObject, handles);
set(get(handles.axes_data_processing2,'Children'),'Visible','off');
set(handles.axes_data_processing,'Visible','off');
set(get(handles.axes_data_processing,'Children'),'Visible','off');
tempstring = '0';          sep= '|';          Numtotal = 0;
for i=1:1:length(handles.furtherprocessingarray)
        if(handles.currentprocessingarray.periodtype ==    …
        handles.furtherprocessingarray(i).periodtype)
                Numtotal = Numtotal + 1;
                if(Numtotal == 1)
                        tempstring = num2str(Numtotal);
                else
                        tempstring = [tempstring,sep,num2str(Numtotal)];
                end                end
 end
set(handles.parameter2_input,'Style','popupmenu');
set(handles.parameter2_input,'String',tempstring);
if(handles.currentprocessingarray.periodnum == 0)
        set(handles.parameter2_input,'Value',1);
else
set(handles.parameter2_input,'Value',handles.currentprocessingarray.periodnum);
```

```
    end
    set(handles.pushbutton_export,'Visible','on');
    set(handles.pushbutton_apply,'Visible','on');
  switch get(handles.parameter3_input,'Value');
    case 1
        if(length(handles.currentprocessingarray.Originaldata)>0)
            set(handles.pushbutton_export,'Enable','on');
            set(handles.pushbutton_apply,'Enable','on');
        else
            set(handles.pushbutton_apply,'Enable','off');
            set(handles.pushbutton_export,'Enable','off');
        end
    case 2
        if(length(handles.currentprocessingarray.reducedata)>0)
            set(handles.pushbutton_export,'Enable','on');
            set(handles.pushbutton_apply,'Enable','on');
        else
            set(handles.pushbutton_apply,'Enable','off');
            set(handles.pushbutton_export,'Enable','off');
        end
     case 3
        if(length(handles.currentprocessingarray.denoisedata)>0)
            set(handles.pushbutton_export,'Enable','on');
            set(handles.pushbutton_apply,'Enable','on');
        else
            set(handles.pushbutton_apply,'Enable','off');
            set(handles.pushbutton_export,'Enable','off');
        end
    end
end
guidata(hObject, handles);
% ---------------------------------------------------------------------
function parameter1_input_KeyPressFcn(hObject, eventdata, handles)
function parameter2_input_Callback(hObject, eventdata, handles)
```

```
if(handles.eventnum == 10)
        set(get(handles.axes_data_processing2,'Children'),'Visible','off');
        set(handles.axes_data_processing,'Visible','off');
        set(get(handles.axes_data_processing,'Children'),'Visible','off');
end
guidata(hObject, handles);

% --------------------------------------------------------------------
function pushbutton_apply_Callback(hObject, eventdata, handles) %#ok<INUSL>
tic
handles.figurecontrol = 0;
[state outhandles]= DataProcessingDataManageeps(handles);
if(state == 1)            handles = outhandles;    end
guidata(hObject, handles);
toc      tdata = toc      tic
[state outhandles] = UpdateDataProcessingInterfaceeps(handles);
if(state == 1)          handles = outhandles; end
guidata(hObject, handles);
toc      tinterface = toc
% --------------------------------------------------------------------
function pushbutton_export_Callback(hObject, eventdata, handles)
switch handles.eventnum
      case 1
         if(length(handles.OriginalPressure.OriginalData) > 0)
                  [filename, pathname] = uiputfile('*.dat', 'write data to file');
                  if isequal(filename,0) | isequal(pathname,0) %#ok<OR2>
                        disp('User selected Cancel');
                  else
                        disp(['User selected',fullfile(pathname,filename)]);
                        flag = writestruct(filename,pathname,handles.OriginalPressure);
                  end
            end
      case 2
            if(length(handles.InterpolationData.Data) > 0)
```

```
            [filename, pathname] = uiputfile('*.dat', 'write data to file');
            if isequal(filename,0) | isequal(pathname,0) %#ok<OR2>
                disp('User selected Cancel');
            else
                disp(['User selected',fullfile(pathname,filename)]);
                flag = writestruct(filename,pathname,handles.InterpolationData);
            end
        end
    case 3
        if(length(handles.DecompisitionData.Data) > 0)
            [filename, pathname] = uiputfile('*.dat', 'write data to file');
            if isequal(filename,0) | isequal(pathname,0) %#ok<OR2>
                disp('User selected Cancel');
            else
                flag = writestruct(filename,pathname,handles.DecompisitionData);
            end                end
    case 5
        if(length(handles.OutlierremovalData.Data) > 0)
            [filename, pathname] = uiputfile('*.dat', 'write data to file');
            if isequal(filename,0) | isequal(pathname,0)
                disp('User selected Cancel');
            else
               disp(['User selected',fullfile(pathname,filename)]);
               flag = writestruct(filename,pathname,handles.OutlierremovalData);
            end                  end
    case 6
        if(length(handles.buddevent.Data) > 0)
            [filename, pathname] = uiputfile('*.dat', 'write data to file');
            if isequal(filename,0) | isequal(pathname,0) %#ok<OR2>
                disp('User selected Cancel');
            else
                disp(['User selected',fullfile(pathname,filename)]);
                flag = writestruct(filename,pathname,handles.buddevent);
            end                end
```

```
    case 7
        if length(handles.recoverrate.Data) > 0
            [filename, pathname] = uiputfile('*.dat', 'write data to file');
            if isequal(filename,0) | isequal(pathname,0) %#ok<OR2>
                disp('User selected Cancel');
            else
                disp(['User selected',fullfile(pathname,filename)]);
                flag = writestruct(filename,pathname,handles.recoverrate);
            end
        end
    case 10
        if length(handles.currentprocessingarray) > 0
            [filename, pathname] = uiputfile('*.dat', 'write data to file');
            if isequal(filename,0) | isequal(pathname,0) %#ok<OR2>
                disp('User selected Cancel');
            else
            disp(['User selected',fullfile(pathname,filename)]);
            flag = writestruct(filename,pathname,handles.currentprocessingarray);
            end                 end
    otherwise
end
% --------------------------------------------------------------------
function menu_figure_Callback(hObject, eventdata, handles)
if(handles.figurecontrol == 1)
    temp = UpdateDataProcessingInterfaceeps(handles);
else
    msgbox('There is no figure to plot!','icon');
end
handles.figurecontrol = 0;          guidata(hObject, handles);
```

Data managment function

```
% --------------------------------------------------------------------
%          This function is used to manage the data.
%        Version: 1.1.0
```

```
%       Author:    Xiaogang Li
%       Date:      May-2007
function    [state varargout] = DataProcessingDataManage(handles)
      state = 1;          updatedhandles = handles;
      eventnum = handles.eventnum;
      switch eventnum
            case -1
            handles.OriginalPressure = struct('Num',0,'Name', {''},'OriginalData',[]…
            ,'Preprocessingdata',[]);
            handles.InterpolationData = struct('Num',1,'timestep',[10],…
            'method',{'None'}, 'Data',[]);
            handles.OutlierremovalData = struct('Num',4,'Data',[]);
       handles.buddevent = struct('Num',5,'parameter',[0.8 0.02 5],'Data',[],'BUDD',[]);
            handles.recoverrate = struct('Num',6,'Totalrate',[0 0],'Data',[]);
            handles.currentprocessingarray =    struct('Num',0,'periodtype',1,…
            'periodnum',0,Originaldata',[],'reducedata',[],'denoisedata',[]);
            handles.furtherprocessingarray = [];
            case 0
                  handles.CurrentWell = [];                  handles.WellArray = [];
                  handles.OriginalPressure = struct('Num',0,'Name', {''},'OriginalData'…
                  ,[],'Preprocessingdata',[]);
                  handles.InterpolationData = struct('Num',1,'timestep'…
                  ,[10],'method',{'None'}, 'Data',[]);
                  handles.DecompisitionData = struct('Num',2,'Wavelet'…
                  ,{'haar'},'Level',[2], 'Data',[]);
                  handles.EvaluationData = struct('Num',3,'Type',{'hist'},…
                  'Mean',[0],'Stddata',[0],'State',[0]);
                  handles.OutlierremovalData = struct('Num',4,'Data',[]);
                  handles.buddevent = struct('Num',5,'parameter',…
                  [0 0.02 5],'Data',[],'BUDD',[]);
                  handles.recoverrate = struct('Num',6,'Totalrate',[0 0],'Data',[]);
             handles.currentprocessingarray =    struct('Num',0,'periodtype',1,'periodnum'…
                  ,0,'Originaldata',[],'reducedata',[],'denoisedata',[]);
                  handles.furtherprocessingarray = [];
```

```
[FileName,PathName,FilterIndex] = uigetfile({ ...
'*.dat','Signalfile(*.dat)';'*.tpr','Signal file(*.tpr)';'*.*','All file(*.*)'});
handles.FileName = FileName;        handles.PathName = PathName;
handles.SaveFileName =[];           handles.SavePathName = [];
handles.FilterIndex = FilterIndex;
if isequal(FileName,0) | isequal(PathName,0)
msgbox('There is no data file for inputing!','File Information','warn');
    disp('User selected Cancel');
else
    disp(['User selected',fullfile(PathName,FileName)]);
    switch FilterIndex
        case 1
            A = load([PathName FileName]);
            if(size(A,2) == 3)        A = A(:,1:2);          end
            convconst = 0.1450377;
            A(:,2) = A(:,2)*convconst;
            if(length(A)>0)
                [pathstr,name,ext,versn] = fileparts(FileName);
                handles.OriginalPressure.Name = name;
                handles.OriginalPressure.OriginalData    = A(:,1:2);
            end
         case 2
            A = load([PathName FileName]);
            if(length(A)>0)
                [pathstr,name,ext,versn] = fileparts(FileName);
                handles.OriginalPressure.Name = name;
                handles.OriginalPressure.OriginalData = A(:,1:2);
            end
      otherwise
        end
   end
case 1
    OriginalPressure = handles.OriginalPressure.OriginalData;
    handles.figurecontrol = 0;        t = cputime;      temprepeat = [];
```

```
temprepeat = OriginalPressure(1,:);
totallenght = size(OriginalPressure ,1);
totalsummary = size(OriginalPressure ,1);
oldtime = temprepeat(1,1);                    beginpoint = 0;
while(totallenght>0)
    if size(temprepeat,1)==1
        beginpoint = 2;
    else
        beginpoint = 1;
    end
    negtivebegin = 0; negtiveend = 0;    negtiveflag = 0;
    for(i= beginpoint:1:totallenght)
        newtime = OriginalPressure(i,1);
        if(newtime <= oldtime || OriginalPressure(i,2) <= 0)
            if(negtiveflag == 0)    negtivebegin = i;    end
            negtiveflag = 1;        oldtime = newtime;
        else
            if(negtiveflag == 1)
                negtiveend = i-1;
temprepeat = [temprepeat;OriginalPressure(beginpoint:negtivebegin-1,:)];
OriginalPressure =    OriginalPressure(negtiveend+1:size(OriginalPressure,1),:);
                pourcent = size(temprepeat,1)*100/totalsummary
                negtiveflag = 0;            break;
            end
        end
        if(i== totallenght)
            if(size(temprepeat,1)==1)
                temprepeat = OriginalPressure(1:i,:);
            else
            temprepeat = [temprepeat ; OriginalPressure(1:i,:)];
            end
            OriginalPressure = [];          break;              end
        oldtime = newtime;
    end         totallenght = size(OriginalPressure ,1);
```

```
        end
        switch handles.FilterIndex
            case 2
                temp = handles.OriginalPressure.OriginalData;
                if(temp(length(temp),1)<2)
                    xd = temprepeat(:,2);
                else
                    xd = wden(temprepeat(:,2),'heursure','s','one',20,'db10');
                    xd = temprepeat(:,2);
                end
                temprepeat(:,2) = xd;
        end
        handles.OriginalPressure.Preprocessingdata = temprepeat;
    case 2
        OriginalPressure = handles.OriginalPressure;
        InterpolationData = handles.InterpolationData;
        handles.figurecontrol = 0;
        timestepstring = get(handles.parameter2_input,'String');
        if(~strcmp('None',timestepstring))
            InterpolationData.timestep = str2num(timestepstring);
        end
        temporiginaldata = OriginalPressure.Preprocessingdata;
        xi = temporiginaldata(1,1):InterpolationData.timestep/3600: ...
            temporiginaldata(length(temporiginaldata),1);
        intermethod = {'nearest';'linear';'spline';'pchip'};
        temp = get(handles.parameter1_input,'Value');
        stringmethod = intermethod{temp};
        InterpolationData.method = stringmethod;
        yi = interp1(OriginalPressure.Preprocessingdata…
        (:,1),OriginalPressure.Preprocessingdata(:,2),xi,stringmethod);
        InterpolationData.Data = [xi;yi]';
        handles.InterpolationData    = InterpolationData ;
    case 3
        InterpolationData = handles.InterpolationData;
```

```
        DecompisitionData = handles.DecompisitionData;
        Levelnumber = get(handles.parameter2_input,'String');
        if(~strcmp('None',Levelnumber ))
            DecompisitionData.Level = str2num(Levelnumber);
        end
        intermethod = {'haar';'db1'};
        temp = get(handles.parameter1_input,'Value');
        wstring = intermethod{temp};
        DecompisitionData.Wavelet = wstring;                    lx = 0;
        lx = length(InterpolationData.Data);
        waveletstring =    DecompisitionData.Wavelet;
        [Lo_D,Hi_D] = wfilters(waveletstring,'d');
        lf = length(Lo_D);                      shift = 0;
        first = 2 - shift;                      dwtEXTM = 'sym';
        flagPer = isequal(dwtEXTM,'per');
        if ~flagPer
            lenEXT = lf-1; last = lx+lf-1;
        else
            lenEXT = lf/2; last = 2*ceil(lx/2);
        end
      yi = InterpolationData.Data; y = wextend('1D',dwtEXTM,yi(:,2),lenEXT);
        z = wconv1(y,Hi_D,'valid');             d = z(first:1:last);
        temp = InterpolationData.Data;      detail = [temp(:,1) d];
        DecompisitionData.Data = detail;
        handles.DecompisitionData    =    DecompisitionData;
    case 4
        if length(handles.DecompisitionData.Data)> 0
            DetailData = handles.DecompisitionData.Data;
            handles.EvaluationData.Mean = trimmean(DetailData(:,2),10);
            %handles.EvaluationData.Stddata = iqr(DetailData(:,2));
            handles.EvaluationData.Stddata = std(DetailData(:,2));
            handles.EvaluationData.State = 1;
            handles.EvaluationData.Type = 'hist';
        end
```

```
        case 5
            if(handles.EvaluationData.State == 1 && handles.FilterIndex==1)
                temp = handles.InterpolationData.Data;
                tt= outlierremovalfeps(temp(:,2));
                if(size(tt,1) ~= size(temp,1))
                    temp(:,2) = tt(2:length(tt));
                else
                    temp(:,2) = tt;
                end
                handles.OutlierremovalData.Data = temp;
                if(length(handles.WellArray)>0 && length(test)>0)
          handles.CurrentWell = set(handles.CurrentWell,'dataprocesspressure',test);
                    string_wellname = get(handles.CurrentWell,'WellName');
                        numberwell = length(handles.WellArray);
                        if(numberwell == 0)
                                handles.WellArray{1} = handles.CurrentWell;
                        else
                                for(i=1:1:numberwell)
if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
                                 handles.WellArray{i} = handles.CurrentWell;
                                        break;
                                    else
                                        if(i==numberwell)
                                handles.WellArray{i+1} = handles.CurrentWell;
                                        end
                                    end          end          end
                end
            else
                temp = handles.InterpolationData.Data;
                handles.OutlierremovalData.Data = temp;
                if(length(handles.WellArray)>0 && length(test)>0)
          handles.CurrentWell = set(handles.CurrentWell,'dataprocesspressure',test);
                    %save recoverrate to current well
                    string_wellname = get(handles.CurrentWell,'WellName');
```

```
                numberwell = length(handles.WellArray);
                if(numberwell == 0)
                        handles.WellArray{1} = handles.CurrentWell;
                else
                        for(i=1:1:numberwell)
if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
                handles.WellArray{i} = handles.CurrentWell;
                                break;
                            else
                                if(i==numberwell)
                handles.WellArray{i+1} = handles.CurrentWell;
            end   end   end   end   end
        end
    case 6
        if(length(handles.OutlierremovalData.Data)> 1 & …
            length(handles.InterpolationData.Data)>1)
            handles.currentprocessingarray = struct('Num',0,'periodtype',1,…
            'periodnum',0,'Originaldata',[],'reducedata',[],'denoisedata',[]);
            handles.furtherprocessingarray = [];
            switch   handles.FilterIndex
              case 1
                    pressuredata = handles.OutlierremovalData.Data;
                  xd = wden(pressuredata(:,2),'heursure','s','one',15,'sym8');
                    pressuredata(:,2) = xd;              x = pressuredata;
                    m = 1:1:size(x,1);                   x= [m', x];
                 m = 1:1:size(handles.OriginalPressure.Preprocessingdata);
              originalm = [m', handles.OriginalPressure.Preprocessingdata];
                    handles.buddevent.Data = originalm;
                    temp = get(handles.parameter1_input,'String');
                    minthrethord = str2num(temp); %#ok<ST2NM>
                    temp = get(handles.parameter2_input,'String');
                    interval = str2num(temp); %#ok<ST2NM>
                    temp = get(handles.parameter3_input,'String');
                    samplenumber = str2num(temp);
```

```
                parameter = [minthrethord,interval,samplenumber];
                filetypedata = handles.FilterIndex;
               [BUDD] = distinguishBUDD_functioneps(x,originalm,…
                filetypedata,minthrethord,interval,samplenumber);
                x = originalm;
            case 2
              m = 1:1:size(handles.OriginalPressure.Preprocessingdata);
            originalm = [m', handles.OriginalPressure.Preprocessingdata];
                handles.buddevent.Data = originalm;
                temp = get(handles.parameter1_input,'String');
                minthrethord = str2num(temp); %#ok<ST2NM>
                temp = get(handles.parameter2_input,'String');
                interval = str2num(temp); %#ok<ST2NM>
                temp = get(handles.parameter3_input,'String');
                samplenumber = str2num(temp); %#ok<ST2NM>
                parameter = [minthrethord,interval,samplenumber];
                filetypedata = handles.FilterIndex;
                [BUDD] = distinguishBUDD_functionmdteps(…
originalm,originalm,filetypedata,minthrethord,interval,samplenumber);
                x = originalm;                    otherwise
        end
        if(length(BUDD)==0)
            warndlg('Too bigger threshold','!! Warning !!')
        else
            handles.buddevent.BUDD = BUDD;
            handles.buddevent.parameter = parameter;
            tempperiod = struct('Num',0,'periodtype',1,'periodnum',0,...
                'Originaldata',[],'reducedata',[],'denoisedata',[]);
            periodnumdd = 0; periodnumshutin = 0;
            periodnumdrop = 0;
            for i=1:1:size(BUDD,1)
                currentperiod = tempperiod;    currentperiod.Num = i;
                currentperiod.periodtype = BUDD(i,2);
                switch currentperiod.periodtype
```

```
                                case 1
                                    periodnumdd = periodnumdd + 1;
                                    currentperiod.periodnum = periodnumdd;
                                case -1
                                    periodnumshutin = periodnumshutin + 1;
                                    currentperiod.periodnum = periodnumshutin;
                                case -2
                                    periodnumdrop = periodnumdrop + 1;
                                    currentperiod.periodnum = periodnumdrop;
                            end
                            if i==1             low_boundary = 1;
                            else    low_boundary = find(x(:,2) >= BUDD(i,3),1);
                            end
                            if(i==(size(BUDD,1)))
                                high_boundary = size(x,1);       else
                                high_boundary = find(x(:,2) >= BUDD(i,4),1);
                            end
                        currentperiod.Originaldata = x(low_boundary:high_boundary,2:3);
    handles.furtherprocessingarray = [handles.furtherprocessingarray currentperiod];
                    handles.currentprocessingarray = handles.furtherprocessingarray(1);
                            end       end
                end
            case 7
                if(length(handles.buddevent.BUDD) > 1)
                    temp = get(handles.parameter1_input,'String');
                    totaloil = str2num(temp);
                    temp = get(handles.parameter2_input,'String');
                    totalgas = str2num(temp);
                    BUDD = handles.buddevent.BUDD;
                    if(totalgas ==0)      totalgas = 0.1;          end
                    recovered_rate = recover_function(BUDD,totalgas,totaloil);
                    handles.recoverrate.Totalrate = [totaloil totalgas];
                    handles.recoverrate.Data = recovered_rate;
                    lengthdata = length(recovered_rate);
```

```
                tempnewdata = [recovered_rate(:,1) recovered_rate(:,4) …
                recovered_rate(:,5)];
                m =    [recovered_rate(lengthdata,1)+recovered_rate(lengthdata…
                ,2) recovered_rate(lengthdata,4) recovered_rate(lengthdata,5)];
                tempnewdata = [tempnewdata; m];
              if(length(handles.WellArray)>0 && length(recovered_rate)>0)
                    tempWellProduction = handles.CurrentWell.WellProduction;
                    tempWellProduction{4} = tempnewdata;
                     handles.CurrentWell = set(handles.CurrentWell,…
                    'WellProduction',tempWellProduction);
                    %save recoverrate to current well
                    string_wellname = get(handles.CurrentWell,'WellName');
                          numberwell = length(handles.WellArray);
                          if(numberwell == 0)
                                handles.WellArray{1} = handles.CurrentWell;
                          else
                                for i=1:1:numberwell
if(logical(strcmp(string_wellname,get(handles.WellArray{i},'WellName'))))
   handles.WellArray{i} = handles.CurrentWell; break;
                                    else
 if(i==numberwell)    handles.WellArray{i+1} = handles.CurrentWell;
      end     end       end         end         end
      end
case 8
if isequal(handles.FileName,0) || isequal(handles.PathName,0)
          msgbox('There is no data for outputing!','File Information','warn');
          disp('User selected Cancel');
else
          filename = [];                  pathname = [];
          [filename,pathname,FilterIndex] = uiputfile({ ...
                '*.dat','Signal file(*.dat)';'*.*','All file(*.*)'});
          handles.SaveFileName = filename;
          handles.SavePathName = pathname;
   if isequal(handles.SaveFileName,0) || isequal(handles.SavePathName,0)
```

```
                msgbox('There is no file for outputing!','File Information','warn');
                disp('User selected Cancel');
            else
                fid = fopen([handles.SavePathName,handles.SaveFileName],'w');
                        fprintf(fid,'*****************************\n');
                        fprintf(fid,'*\n');
                        fprintf(fid,'*This file is used to …
                         contain the fluid period data. \n');
                        fprintf(fid,'*\n');    fprintf(fid,'*        Date: ');
                        fprintf(fid,'%s\n',date);    fprintf(fid,'*\n');
                        fprintf(fid,'*****************************\n\n');
                        fprintf(fid,'Total Number of period \n');
                    fprintf(fid,'%d\n\n',length(handles.furtherprocessingarray));
                        status = fclose(fid);
                        for i = 1:1:length(handles.furtherprocessingarray)
                            if(i==1)        state = 'a';
                            else    state = 'a';        end
                            writestruct(handles.SaveFileName,handles….
                    SavePathName,handles.furtherprocessingarray(i),state);
                            fid = fopen([handles.SavePath…
                            Name,handles.SaveFileName],'a');
                                tempdata = handles….
                                furtherprocessingarray(i).Originaldata;
                                beginpoint = tempdata(1,2);
                                endpoint = tempdata(size(tempdata,1),2);
                                fprintf(fid,'%s:\n','Beginpoint');
                                fprintf(fid,'%f\n\n',beginpoint);
                                fprintf(fid,'%s:\n','endpoint');
                                fprintf(fid,'%f\n\n',endpoint);
                            status = fclose(fid);
                        end                 end
        end
case 9
    if isempty(handles.SaveFileName) || isempty(handles.SavePathName) ||...
```

```
        isequal(handles.SaveFileName,0) || isequal(handles.SavePathName,0)
            msgbox('There is no file for outputing!','File Information','warn');
            disp('User selected Cancel');
        else
            fid = fopen([handles.SavePathName,handles.SaveFileName],'w');
                fprintf(fid,'**************************\n');
                fprintf(fid,'*\n');          fprintf(fid,'*\n');
                fprintf(fid,'*      Date: ');    fprintf(fid,'%s\n',date);
                fprintf(fid,'*\n'); fprintf(fid,'****************\n\n');
                fprintf(fid,'Total Number of period \n');
              fprintf(fid,'%d\n\n',length(handles.furtherprocessingarray));
                status = fclose(fid);
                for i = 1:1:length(handles.furtherprocessingarray)
                    if(i==1)              state = 'a';
                    else                  state = 'a';          end
                writestruct(handles.SaveFileName,handles….
                SavePathName,handles.furtherprocessingarray(i),state);
          fid = fopen([handles.SavePathName,handles.SaveFileName],'a');
          tempdata      =      handles.furtherprocessingarray(i).Originaldata;
          beginpoint = tempdata(1,2);endpoint = tempdata(size(tempdata,1),2);
          fprintf(fid,'%s:\n','Beginpoint'); fprintf(fid,'%f\n\n',beginpoint);
          fprintf(fid,'%s:\n','endpoint');   fprintf(fid,'%f\n\n',endpoint);
          status = fclose(fid);                                  end
        end
case 10
        if(length(handles.furtherprocessingarray) > 0)
        tempdata =   struct('Num',0,'periodtype',0,'periodnum'…
        ,0,'Originaldata',[],'reducedata',[],'denoisedata',[]);
            tempstring = get(handles.parameter2_input,'String');
            tempstring = tempstring(get(handles.parameter2_input,'Value'),:);
            periodnum = str2num(tempstring);
            temp = get(handles.parameter1_input,'Value'); periodtype = 0;
            switch temp
                case 1      periodtype = 1;
```

```
        case 2      periodtype = -1;
        case 3      periodtype = -2;
end
handles.currentprocessingarray = tempdata;
for i=1:1:length(handles.furtherprocessingarray)
        tempdata = handles.furtherprocessingarray(i);
        if(tempdata.periodtype == periodtype)…
           && (tempdata.periodnum == periodnum)
              handles.currentprocessingarray = tempdata;    break;
        else
        tempdata =    struct('Num',0,'periodtype',0,'periodnum',0,…
        'Originaldata',[],'reducedata',[],'denoisedata',[]);
        end
end
if(handles.currentprocessingarray.Num == 0)
        tempdata.periodtype = periodtype;
        handles.currentprocessingarray = tempdata;
end
orignaldata = handles.currentprocessingarray.Originaldata;
if(get(handles.parameter3_input,'Value') =…
= 2 && length(orignaldata)>0)
denoisedata = wden(orignaldata(:,2),'heursure','s','one',10,'sym8');
        orignaldata(:,2) = denoisedata;
        handles.currentprocessingarray.denoisedata = orignaldata;
        numperiod = handles.currentprocessingarray.Num;
        handles.furtherprocessingarray(numperiod) = …
         handles.currentprocessingarray;
end
if(get(handles.parameter3_input,'Value') == 3 && …
length(handles.currentprocessingarray.denoisedata)>0)
        denoisedata = handles.currentprocessingarray.denoisedata;
        arraycontainer = compressfeps(denoisedata);
        handles.currentprocessingarray.reducedata = arraycontainer;
        numperiod = handles.currentprocessingarray.Num;
```

```
                        handles.furtherprocessingarray(numperiod) = …
                        handles.currentprocessingarray;
                    end     end    otherwise
    end
     if(state == 1)
          updatedhandles = handles;     varargout(1) =    {updatedhandles};
     else
          varargout(1) =    {updatedhandles};
     end
return;
```

update the interface function

```
% ------------------------------------------------------------------
%         This function is used to update the interface.
function    [state varargout] = UpdateDataProcessingInterfaceeps(handles)
state    = 1;       OriginalPressure = handles.OriginalPressure;
if(length(OriginalPressure.OriginalData) <= 0)
     isclass = 1;
     set(handles.axes_data_processing,'Visible','off');
     set(get(handles.axes_data_processing,'Children'),'Visible','off');
     h = findobj('Tag','uipanel_main_control');
     set(h,'Visible','off');         hchildren = get(h,'Children');
     for(i=1:1:length(hchildren))              set(hchildren(i),'Visible','off');         end
     h = findobj('Tag','Menu_data_processing');         set(h,'Enable','off');
     h = findobj('Tag','menu_save_signal');              set(h,'Enable','off');
     h = findobj('Tag','Menu_save_as');                  set(h,'Enable','off');
     h = findobj('Tag','Menu_exit');                     set(h,'Enable','off');
     if(state == 1)
          updatedhandles = handles;    varargout(1) =    {updatedhandles};
      else    varargout(1) =    {updatedhandles};       end          return;
else
    set(handles.Menu_data_processing,'Enable','on');
    set(handles.menu_save_signal,'Enable','on');
    set(handles.Menu_save_as,'Enable','on');
```

```
    set(handles.Menu_exit,'Enable','on');
    set(handles.Menu_data_processing,'Visible','on');
    hchildren = get(handles.Menu_data_processing,'Children');
    for(i=1:1:length(hchildren))
        set(hchildren(i),'Visible','on');          set(hchildren(i),'Enable','off');
    end
  if(length(OriginalPressure.OriginalData) > 0 & length(OriginalPressure.Name) > 0)
        set(handles.Menu_data_interpolation ,'Enable','on');
        set(handles.Menu_data_interpolation,'Checked','off');
if(handles.eventnum ==2) set(handles.Menu_data_interpolation,'Checked','on'); end
    end
    if(length(handles.InterpolationData.Data) > 0 )
        set(handles.Menu_data_decomposition,'Enable','on');
        set(handles.Menu_data_decomposition,'Checked','off');
        if(handles.eventnum ==3)
            set(handles.Menu_data_decomposition,'Checked','on');
        end
    end
    if(length(handles.DecompisitionData.Data) > 0 )
        set(handles.Menu_data_evaluation,'Enable','on');
        set(handles.Menu_data_evaluation,'Checked','off');
        if(handles.eventnum ==4)
            set(handles.Menu_data_evaluation,'Checked','on');
        end
    end
    if(handles.EvaluationData.State == 1)
        set(handles.Menu_outlier_removal,'Enable','on');
        set(handles.Menu_outlier_removal,'Checked','off');
        if(handles.eventnum ==5)
            set(handles.Menu_outlier_removal,'Checked','on');
        end
    end
    if(length(handles.OutlierremovalData.Data) > 1)
        set(handles.Menu_identify,'Enable','on');
```

```
        set(handles.Menu_identify,'Checked','off');
        if(handles.eventnum ==6)
            set(handles.Menu_identify,'Checked','on');
        end
    end
    if(length(handles.buddevent.Data) > 1)
        set(handles.Menu_recovery_rate,'Enable','on');
        set(handles.Menu_recovery_rate,'Checked','off');
        if(handles.eventnum ==7)
            set(handles.Menu_recovery_rate,'Checked','on');
        end
    end
    legend('off');
    switch handles.eventnum
        case 0
            switch handles.figurecontrol
                case 0
                    set(handles.axes_data_processing,'Visible','on');
                    set(get(handles.axes_data_processing,'Children'),'Visible','on');
                    axes(handles.axes_data_processing);
                    set(handles.axes_data_processing2,'Visible','off');
                   set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                    handles.figurecontrol = 1;
                case 1
                    cla;    set(handles.axes_data_processing,'Visible','off');
                    set(handles.axes_data_processing2,'Visible','off');
                    figure;    handles.figurecontrol = 0;
            end
           temp= OriginalPressure.OriginalData; plot(temp(:,1),temp(:,2),'*');
            set(get(handles.axes_data_processing,'Xlabel'),'String',
           …'Time(hours)','FontWeight','bold','FontSize',12);
            set(get(handles.axes_data_processing,'Ylabel'),'String',
           …'Pressure(psi)','FontWeight','bold','FontSize',12);
            title('Original PDG Pressure','FontWeight','bold','FontSize',14);
```

```
        legend('on');           legend('Original Data');
case 1
        if(length(OriginalPressure.Preprocessingdata) > 0)
              switch handles.figurecontrol
                case 0
                 set(handles.axes_data_processing,'Visible','on');
                 set(get(handles.axes_data_processing,'Children'),'Visible','on');
                 axes(handles.axes_data_processing);
                 set(handles.axes_data_processing2,'Visible','off');
                 set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                 handles.figurecontrol = 1;temp = OriginalPressure.OriginalData;
                 plot(temp(:,1),temp(:,2),'*-');t
                temp = OriginalPressure.Preprocessingdata; hold on;
                plot(temp(:,1),temp(:,2),'r.-');
                set(get(handles.axes_data_processing,'Xlabel'),'String','
                      …Time(hours)','FontWeight','bold','FontSize',12);
                          set(get(handles.axes_data_processing,'Ylabel'),'String','
                      …Pressure(psi)','FontWeight','bold','FontSize',12);
                 title('Original PDG Pressure','FontWeight','bold','FontSize',14);
                 legend('on');    legend('Original Data','Preprocessing result');
case 2
        if(length(handles.InterpolationData.Data) > 0)
                 switch handles.figurecontrol
                 case 0
                 set(handles.axes_data_processing,'Visible','on');
                 set(get(handles.axes_data_processing,'Children'),'Visible','on');
                 axes(handles.axes_data_processing);
                 set(handles.axes_data_processing2,'Visible','off');
                 set(get(handles.axes_data_processing2,'Children'),'Visible','off')
                 temp = handles.InterpolationData.Data;
                 plot(temp(:,1),temp(:,2),'-or');                hold on;
                 temp = handles.OriginalPressure.Preprocessingdata;
                 plot(temp(:,1),temp(:,2),'.-b');
                 set(get(handles.axes_data_processing,'Xlabel'),'String',
```

```
                    …'Time(hours)','FontWeight','bold','FontSize',12);
                set(get(handles.axes_data_processing,'Ylabel'),'String',
                    …'Pressure(psi)','FontWeight','bold','FontSize',12);
                title('Original PDG Pressure(interpolation)',
                    …'FontWeight','bold','FontSize',14);
                legend('on');legend('Interpolation Data','Original Data');
                hold off;                handles.figurecontrol = 1;
                    case 1      cla;
                        set(handles.axes_data_processing,'Visible','off');
                        set(handles.axes_data_processing2,'Visible','off');
                        figure; temp = handles.InterpolationData.Data;
                        plot(temp(:,1),temp(:,2),'-or');    hold on;
                        temp = handles.OriginalPressure.Preprocessingdata;
                      plot(temp(:,1),temp(:,2),'.-b');
                      Xlabel('Time(hours)','FontWeight','bold','FontSize',12);
                      Ylabel('Pressure(psi)','FontWeight','bold','FontSize',12);
        title('Original PDGressure(interpolation)','FontWeight','bold','FontSize',14);
                      legend('on');legend('Interpolation Data','Original Data');
                      hold off;      handles.figurecontrol = 0;
                end
            else
                cla; set(handles.axes_data_processing,'Visible','off');
                set(handles.axes_data_processing2,'Visible','off');
      title('Original PDG Pressure(interpolation)','FontWeight','bold','FontSize',14);
            end
        case 3
            if(length(handles.DecompisitionData.Data) > 0)
                switch handles.figurecontrol
                    case 0
set(handles.axes_data_processing,'Visible','on');      box off;
tempscale = handles.DecompisitionData.Data;
temppressure = handles.InterpolationData.Data;
ax1 = handles.axes_data_processing;
hl1 = line(temppressure(:,1),temppressure(:,2),'Color','r');
```

```
    set(ax1,'XColor','r','YColor','r');
    set(get(ax1,'Ylabel'),'String','Pressure(psi)','FontWeight','bold','FontSize',12)
    set(get(ax1,'Xlabel'),'String','Time(hours)','FontWeight','bold','FontSize',12);
    title({'Signal and Sub-Signal';''} ,'FontWeight','bold','FontSize',14);
    set(handles.axes_data_processing2,'Visible','on');
    hl2 = line(tempscale(:,1),tempscale(:,2),'Color','k','Parent',ax2);
    handles.figurecontrol = 1;
case 1
    cla;
        set(handles.axes_data_processing,'Visible','off');
        set(get(handles.axes_data_processing2,'Children'),'Visible','off');
        set(handles.axes_data_processing2,'Visible','off');
        tempscale = handles.DecompisitionData.Data;
        temppressure = handles.InterpolationData.Data;
        figure;
        [AX,H1,H2] = plotyy(temppressure(:,1)
          …,temppressure(:,2),tempscale(:,1),tempscale(:,2));
          axes(AX(1));xlabel('Time(hours)','FontWeight','bold','FontSize',12);
                ylabel('Pressure(psi)','FontWeight','bold','FontSize',12);    axes(AX(2));
                xlabel('Time(hours)','FontWeight','bold','FontSize',12);
                ylabel('Scale','FontWeight','bold','FontSize',12);
                title(' Signal and Sub-Signal','FontWeight','bold','FontSize',14);
                handles.figurecontrol = 0;
                    end
               else
                     cla;              set(handles.axes_data_processing,'Visible','off');
                     set(handles.axes_data_processing2,'Visible','off');
               end
          case 4
               if handles.EvaluationData.State    == 1
                    switch handles.figurecontrol
                         case 0
                              handles.figurecontrol = 1;
                              cla; set(handles.axes_data_processing2,'Visible','off');
```

```
                    set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                        set(handles.axes_data_processing,'Visible','on');
                    case 1
                        cla;    set(handles.axes_data_processing,'Visible','off');
                        set(handles.axes_data_processing2,'Visible','off');
                    set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                        handles.figurecontrol = 0;          figure;
                end
                  temp = handles.DecompisitionData.Data;
                  detaildata = temp(:,2);
                intermethod = {'hist';'probability density';'Cumulative distribution'};
              switch    handles.EvaluationData.Type
                case 'hist'
                    hist(detaildata,100);
                    title('the distribution of data values','FontWeight','bold','FontSize',14);
            case 'probability density'
                    [f,x,u] = ksdensity(detaildata);
                   title('Density estimate for detaildata', 'FontWeight','bold','FontSize',14)
                   hold on;          [f,x] = ksdensity(detaildata,'width',u/4);
                   plot(x,f,'r');      hold off;
            case 'Cumulative distribution'
                    cdfplot(detaildata);
        title('Cumulative distribution for detaildata', 'FontWeight','bold','FontSize',14)
                    grid off;                       otherwise       end
                else
                        cla;      set(handles.axes_data_processing,'Visible','off');
                        set(handles.axes_data_processing2,'Visible','off');
                        set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                end
        case 5
                 if length(handles.OutlierremovalData.Data)    > 1
                    switch handles.figurecontrol
                        case 0
                            handles.figurecontrol = 1;      cla;
```

```
                    set(handles.axes_data_processing2,'Visible','off');
                set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                    set(handles.axes_data_processing,'Visible','on');
                 case 1
                    set(handles.axes_data_processing,'Visible','off');
                    set(handles.axes_data_processing2,'Visible','off');
                set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                    handles.figurecontrol = 0;    figure;
                end
                 olddata = handles.InterpolationData.Data;
                 newdata = handles.OutlierremovalData.Data;
                 plot(olddata(:,1),olddata(:,2),'o');          hold on;
                 plot(newdata (:,1),newdata (:,2),'*r');
                 title('Original Signal and Removal Outlier Signal','FontSize',14);
                 xlabel('Time(hours)','FontSize',14);
                 ylabel('Pressure(psi)','FontSize',14);              legend('on');
                 legend('Original Data','Removal Outlier Signal');      hold off;
            else          cla;
                 set(handles.axes_data_processing,'Visible','off');
                 set(handles.axes_data_processing2,'Visible','off');
                 set(get(handles.axes_data_processing2,'Children'),'Visible','off');
            end
        case 6
            if length(handles.buddevent.Data)    > 1
                switch handles.figurecontrol
                    case 0
                    handles.figurecontrol = 1;          cla;
                    set(handles.axes_data_processing2,'Visible','off');
                    set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                    set(handles.axes_data_processing,'Visible','on');
                    case 1
                        cla        set(handles.axes_data_processing,'Visible','off');
                        set(handles.axes_data_processing2,'Visible','off');
                        handles.figurecontrol = 0;
```

```
                figure;
            end
            Pressuredata = handles.OriginalPressure.Preprocessingdata;
            plot(Pressuredata(:,1),Pressuredata(:,2),'.-k');        hold on;
            Pressuredata = handles.buddevent.Data;
            BUDD = handles.buddevent.BUDD;
            bdddlength = size(BUDD,1);
            if(bdddlength>0)
                for(i=1:1:bdddlength-1)
                    bingpoint =    find(Pressuredata(:,2) >= BUDD(i,3), 1);
                    endpoint =    find(Pressuredata(:,2) >= BUDD(i,4), 1);
                    if(mod(i,2) == 1)    hold on;        end
            bingpoint =    find(Pressuredata(:,2) >= BUDD(bdddlength,3), 1);
                    endpoint =    length(Pressuredata);
plot(Pressuredata(bingpoint:endpoint,2),Pressuredata(bingpoint:endpoint,3),'g*');
                title('Separated BU and DD','FontSize',14);
                xlabel('Time(Hours)','FontSize',14);
                ylabel('Pressure(psi)','FontSize',14);            hold off;
            else
                cla;    set(handles.axes_data_processing,'Visible','off');
                set(handles.axes_data_processing2,'Visible','off');
            set(get(handles.axes_data_processing2,'Children'),'Visible','off');
            end
        else    cla;
            set(handles.axes_data_processing,'Visible','off');
            set(handles.axes_data_processing2,'Visible','off');
            set(get(handles.axes_data_processing2,'Children'),'Visible','off');
        end
    case 7
        if length(handles.recoverrate.Data)    > 1
            cla;
            set(handles.axes_data_processing2,'Visible','off');
            set(get(handles.axes_data_processing2,'Children'),'Visible','off');
            set(handles.axes_data_processing,'Visible','on');
```

```
                recoverrate = handles.recoverrate.Data;
                hdaygas = stairs(recoverrate(:,1),recoverrate(:,5),'r*-');
                xlabel('Time(hours)','FontSize',14,'color','k');
                ylabel('Rate(bbl/day)','FontSize',14,'color','k');
                title('Recovered Production History','FontSize',14);
            else
                cla;
                set(handles.axes_data_processing,'Visible','off');
                set(handles.axes_data_processing2,'Visible','off');
                set(get(handles.axes_data_processing2,'Children'),'Visible','off');
            end
        case 10
            if handles.currentprocessingarray.Num > 0
                switch handles.figurecontrol
                    case 0
                        handles.figurecontrol = 1;          cla;
                        set(handles.axes_data_processing2,'Visible','off');
                    set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                        set(handles.axes_data_processing,'Visible','on');
                    case 1
                        cla;      set(handles.axes_data_processing,'Visible','off');
                        set(handles.axes_data_processing2,'Visible','off');
                    set(get(handles.axes_data_processing2,'Children'),'Visible','off');
                        handles.figurecontrol = 0;    figure;
                end
                 switch get(handles.parameter3_input,'Value')
                     case   1
                        data =handles.currentprocessingarray.Originaldata;
                        if(length(data)>0)
                            plot(data(:,1),data(:,2),'r*-');
                            xlabel('Time(hours)','FontSize',14,'color','k');
                            ylabel('Pressure(psi)','FontSize',14,'color','k');
                            title('Original Pressure','FontSize',14);
                        end
```

```
                    case    3
                        data =handles.currentprocessingarray.reducedata;
                        if(length(data)>0)
                            plot(data(:,1),data(:,2),'r*');
                            xlabel('Time(hours)','FontSize',14,'color','k');
                            ylabel('Pressure(psi)','FontSize',14,'color','k');
                            title('Compressed Pressure','FontSize',14);
                        end
                    case 2
                        data =handles.currentprocessingarray.denoisedata;
                        if(length(data)>0)
                            hold on;
                            data =handles.currentprocessingarray.Originaldata;
                            plot(data(:,1),data(:,2),'go-');
                            data =handles.currentprocessingarray.denoisedata;
                            plot(data(:,1),data(:,2),'r.-');
                            xlabel('Time(hours)','FontSize',14,'color','k');
                            ylabel('Pressure(psi)','FontSize',14,'color','k');
                            title('Denoised Pressure','FontSize',14);
                            hold off;
                        end
                end
            else
                cla;
                set(handles.axes_data_processing,'Visible','off');
                set(handles.axes_data_processing2,'Visible','off');
                set(get(handles.axes_data_processing2,'Children'),'Visible','off');
            end         otherwise
end
h = findobj('Tag','uipanel_main_control');      set(h,'Visible','on');
hchildren = get(h,'Children');
for(i=1:1:length(hchildren))    set(hchildren(i),'Visible','on');    end
set(handles.uipanel_data_processing,'Visible','on');
hchildren = get(handles.uipanel_data_processing,'Children');
```

```
    for(i=1:1:length(hchildren))
        set(hchildren(i),'Visible','on');    set(hchildren(i),'Enable','off');
    end
   if(length(OriginalPressure.OriginalData) > 0 & length(OriginalPressure.Name) > 0)
        set(handles.pushbutton_data_preprocessing ,'Enable','on');
        set(handles.pushbutton_data_preprocessing ,'ForegroundColor', [0 0 0]);
    end
    if(length(OriginalPressure.Preprocessingdata) > 0 &length(OriginalPressure.Name)
> 0)
        set(handles.pushbutton_data_interpolation ,'Enable','on');
        set(handles.pushbutton_data_interpolation ,'ForegroundColor', [0 0 0]);
    end
    if(length(handles.InterpolationData.Data) > 0 )
        set(handles.pushbutton_data_decomposition ,'Enable','on');
        set(handles.pushbutton_data_decomposition ,'ForegroundColor', [0 0 0]);
    end
    if(length(handles.DecompisitionData.Data) > 0 )
        set(handles.pushbutton_data_evaluation,'Enable','on');
        set(handles.pushbutton_data_evaluation,'ForegroundColor', [0 0 0]);
    end
    if(handles.EvaluationData.State == 1)
        set(handles.pushbutton_outlier_removal,'Enable','on');
        set(handles.pushbutton_outlier_removal,'ForegroundColor', [0 0 0]);
    end
    if(length(handles.OutlierremovalData.Data) > 1)
        set(handles.pushbutton_identify_event,'Enable','on');
        set(handles.pushbutton_identify_event,'ForegroundColor', [0 0 0]);
    end
    if(length(handles.furtherprocessingarray) > 1)
        set(handles.pushbutton_recovering_rate,'Enable','on');
        set(handles.pushbutton_recovering_rate,'ForegroundColor', [0 0 0]);
    end
    set(handles.uipanel_data_information,'Visible','on');
    hchildren = get(handles.uipanel_data_information,'Children');
```

```
    for(i=1:1:length(hchildren))
        set(hchildren(i),'Visible','on'); set(hchildren(i),'Enable','off');
    end
   if(length(OriginalPressure.OriginalData) > 0 & length(OriginalPressure.Name) > 0)
     set(handles.edit_data_name,'Enable','on');set(handles.edit_data_size,'Enable','on');
     set(handles.text_data_name,'Enable','on');set(handles.text_data_size,'Enable','on');
     set(handles.edit_data_name,'String',OriginalPressure.Name);
     set(handles.edit_data_size,'String',num2str(length(OriginalPressure.OriginalData)))
     ;
    end

    switch handles.eventnum
     case 0
          set(handles.uipanel_parameter,'Visible','off');
          hchildren = get(handles.uipanel_parameter,'Children');
          for(i=1:1:length(hchildren))
               set(hchildren(i),'Visible','off');      set(hchildren(i),'Enable','off');
          end
    case 1
          OriginalPressure = handles.OriginalPressure;
          set(handles.pushbutton_data_preprocessing,'ForegroundColor', [1 0 0]);
          set(handles.uipanel_parameter,'Visible','on');
          hchildren = get(handles.uipanel_parameter,'Children');
          for(i=1:1:length(hchildren))
               set(hchildren(i),'Visible','off');      set(hchildren(i),'Enable','off');
          end
          % set apply and export button 'on'
          set(handles.pushbutton_apply,'Enable','on');
          set(handles.pushbutton_apply,'Visible','on');
          set(handles.pushbutton_export,'Visible','on');
         if(length( OriginalPressure.Preprocessingdata)>0)
              set(handles.pushbutton_export,'Enable','on');
         else
              set(handles.pushbutton_export,'Enable','off');
```

```
        end
    case 2
        InterpolationData = handles.InterpolationData;
        set(handles.pushbutton_data_interpolation ,'ForegroundColor', [1 0 0]);
        set(handles.parameter1_name,'Enable','on');
        set(handles.parameter1_name,'Visible','on');
        set(handles.parameter2_name,'Enable','on');
        set(handles.parameter2_name,'Visible','on');
        set(handles.parameter3_name,'Visible','off');
        set(handles.parameter1_input,'Enable','on');
        set(handles.parameter1_input,'Visible','on');
        set(handles.parameter2_input,'Enable','on');
        set(handles.parameter2_input,'Visible','on');
        set(handles.parameter3_input,'Visible','off');
        set(handles.pushbutton_apply,'Enable','on');
        set(handles.pushbutton_apply,'Visible','on');
        set(handles.pushbutton_export,'Visible','on');
        if(length(InterpolationData.Data)>0)
            set(handles.pushbutton_export,'Enable','on');
        else
            set(handles.pushbutton_export,'Enable','off');
        end
        set(handles.parameter1_name,'String','Interpolation Method');
        set(handles.parameter1_input,'Style','popupmenu');
        intermethod = {'nearest';'linear';'spline';'pchip'};            sep = '|';
        stringintermethod =    intermethod{1};
        for i = 2:1:length(intermethod)
           temp = intermethod{i}; stringintermethod = [stringintermethod sep temp];
        end
        set(handles.parameter1_input,'String',stringintermethod);
        valueintermethod = 1;
        for(i = 1:1:length(intermethod))
            temp = intermethod{i};
            if(strcmp(InterpolationData.method,temp))
```

```
                valueintermethod = i;                break;
            end
        end
        set(handles.parameter1_input,'Value',valueintermethod);
        set(handles.parameter2_name,'String','TimeStep(second)');
        set(handles.parameter2_input,'Style','edit');
        set(handles.parameter2_input,'String',num2str(InterpolationData.timestep));
    case 3
        DecompisitionData = handles.DecompisitionData;
        set(handles.pushbutton_data_decomposition ,'ForegroundColor', [1 0 0]);
        set(handles.parameter1_name,'Enable','on');
        set(handles.parameter1_name,'Visible','on');
        set(handles.parameter2_name,'Enable','on');
        set(handles.parameter2_name,'Visible','on');
        set(handles.parameter3_name,'Visible','off');
        set(handles.parameter1_input,'Enable','on');
        set(handles.parameter1_input,'Visible','on');
        set(handles.parameter2_input,'Enable','on');
        set(handles.parameter2_input,'Visible','on');
        set(handles.parameter3_input,'Visible','off');
        set(handles.pushbutton_apply,'Enable','on');
        set(handles.pushbutton_apply,'Visible','on');
        set(handles.pushbutton_export,'Visible','on');
        if(length(DecompisitionData.Data)>0)
            set(handles.pushbutton_export,'Enable','on');
        else
            set(handles.pushbutton_export,'Enable','off');
        end
        set(handles.parameter1_name,'String','Wavelet');
        set(handles.parameter1_input,'Style','popupmenu');
        intermethod = {'haar';'db1'};            sep = '|';
        stringintermethod =    intermethod{1};
        for i = 2:1:length(intermethod)
            temp = intermethod{i}; stringintermethod = [stringintermethod sep temp];
```

```
        end
        set(handles.parameter1_input,'String',stringintermethod);
        valueintermethod = 1;
        for(i = 1:1:length(intermethod))
            temp = intermethod{i};
            if(strcmp(DecompisitionData.Wavelet,temp))
                valueintermethod = i;    break;
            end
        end
        set(handles.parameter1_input,'Value',valueintermethod);
        set(handles.parameter2_name,'String','Level');
        set(handles.parameter2_input,'String',num2str(DecompisitionData.Level));
    case 4
        set(handles.pushbutton_data_evaluation,'ForegroundColor', [1 0 0]);
        set(handles.parameter1_name,'Enable','on');
        set(handles.parameter1_name,'Visible','on');
        set(handles.parameter2_name,'Enable','on');
        set(handles.parameter2_name,'Visible','on');
        set(handles.parameter3_name,'Visible','on');
        set(handles.parameter3_name,'Enable','on');
        set(handles.parameter1_input,'Enable','on');
        set(handles.parameter1_input,'Visible','on');
        set(handles.parameter2_input,'Enable','on');
        set(handles.parameter2_input,'Visible','on');
        set(handles.parameter3_input,'Visible','on');
        set(handles.parameter3_input,'Enable','on');
        set(handles.pushbutton_apply,'Enable','off');
        set(handles.pushbutton_apply,'Visible','off');
        set(handles.pushbutton_export,'Enable','off');
        set(handles.pushbutton_export,'Visible','off');
        set(handles.parameter1_name,'String','Probability Visualization');
        set(handles.parameter1_input,'Style','popupmenu');
        intermethod = {'hist';'probability density';'Cumulative distribution'};
        sep = '|';      stringintermethod =    intermethod{1};
```

```
    for i = 2:1:length(intermethod)
      temp = intermethod{i};    stringintermethod = [stringintermethod sep temp];
    end
    set(handles.parameter1_input,'String',stringintermethod);
    valueintermethod = 1;
    for(i = 1:1:length(intermethod))
          temp = intermethod{i};
          if(strcmp(handles.EvaluationData.Type,temp))
                valueintermethod = i; break;
          end
    end
    set(handles.parameter1_input,'Value',valueintermethod);
    set(handles.parameter2_name,'String','Mean Value');
    set(handles.parameter2_input,'Style','edit');
  set(handles.parameter2_input,'String',num2str(handles.EvaluationData.Mean));
    set(handles.parameter3_name,'String','Standard Deviation');
    set(handles.parameter3_input,'Style','edit');
set(handles.parameter3_input,'String',num2str(handles.EvaluationData.Stddata));
case 5
    set(handles.pushbutton_outlier_removal ,'ForegroundColor', [1 0 0]);
    set(handles.uipanel_parameter,'Visible','on');
    hchildren = get(handles.uipanel_parameter,'Children');
    for(i=1:1:length(hchildren))
          set(hchildren(i),'Visible','off'); set(hchildren(i),'Enable','off');
    end
    set(handles.pushbutton_apply,'Enable','on');
    set(handles.pushbutton_apply,'Visible','on');
    set(handles.pushbutton_export,'Visible','on');
  if(length( handles.OutlierremovalData.Data)>0)
          set(handles.pushbutton_export,'Enable','on');
  else
          set(handles.pushbutton_export,'Enable','off');
  end
case 6
```

```
        buddevent = handles.buddevent;
        parameter = buddevent.parameter;
        set(handles.pushbutton_identify_event,'ForegroundColor', [1 0 0]);
        set(handles.parameter1_name,'Enable','on');
        set(handles.parameter1_name,'Visible','on');
        set(handles.parameter2_name,'Enable','on');
        set(handles.parameter3_name,'Enable','on');
        set(handles.parameter2_name,'Visible','on');
        set(handles.parameter3_name,'Visible','on');
        set(handles.parameter1_input,'Enable','on');
        set(handles.parameter1_input,'Visible','on');
        set(handles.parameter2_input,'Enable','on');
        set(handles.parameter3_input,'Enable','on');
        set(handles.parameter2_input,'Visible','on');
        set(handles.parameter3_input,'Visible','on');
        set(handles.pushbutton_apply,'Enable','on');
        set(handles.pushbutton_apply,'Visible','on');
        set(handles.pushbutton_export,'Visible','on');
        if(length(buddevent.Data)>0)
            set(handles.pushbutton_export,'Enable','on');
        else
            set(handles.pushbutton_export,'Enable','off');
        end
        set(handles.parameter1_name,'String','threshold');
        set(handles.parameter1_name,'Style','text');
        set(handles.parameter1_input,'Style','edit');
        set(handles.parameter1_input,'String',num2str(parameter(1)));
        set(handles.parameter2_name,'String','Interval');
        set(handles.parameter2_input,'Style','edit');
        set(handles.parameter2_input,'String',num2str(parameter(2)));
        set(handles.parameter3_name,'String','Sample Number');
        set(handles.parameter3_input,'Style','edit');
        set(handles.parameter3_input,'String',num2str(parameter(3)));
    case 7
```

```
    buddevent = handles.buddevent;
    set(handles.pushbutton_recovering_rate,'ForegroundColor', [1 0 0]);
    set(handles.parameter1_name,'Enable','on');
    set(handles.parameter1_name,'Visible','on');
    set(handles.parameter2_name,'Enable','on');
    set(handles.parameter2_name,'Visible','on');
    set(handles.parameter3_name,'Visible','off');
    set(handles.parameter1_input,'Enable','on');
    set(handles.parameter1_input,'Visible','on');
    set(handles.parameter2_input,'Enable','on');
    set(handles.parameter2_input,'Visible','on');
    set(handles.parameter3_input,'Visible','off');
    set(handles.pushbutton_apply,'Enable','on');
    set(handles.pushbutton_apply,'Visible','on');
    set(handles.pushbutton_export,'Visible','on');
    if(length(recoverrate.Data)>0)
        set(handles.pushbutton_export,'Enable','on');
    else
        set(handles.pushbutton_export,'Enable','off');
    end
    set(handles.parameter1_name,'String','Totaloil(bbl)');
    set(handles.parameter1_name,'Style','text');
    set(handles.parameter1_input,'Style','edit');
    if(parameter(1) == 0.1)
        parameter(1)    = 0;
    end
    set(handles.parameter1_input,'String',num2str(parameter(1)));
    if(parameter(2) == 0.1)
        parameter(2)    = 0;
    end
    set(handles.parameter2_name,'String','TotalGas(mmscf)');
    set(handles.parameter2_input,'Style','edit');
    set(handles.parameter2_input,'String',num2str(parameter(2)));
case 10
```

```
currentprocessingarray = handles.currentprocessingarray;
furtherprocessingarray = handles.furtherprocessingarray;
if(length(furtherprocessingarray)>0 & currentprocessingarray.Num >=0)
    set(handles.pushbutton_recovering_rate,'ForegroundColor', [1 0 0]);
    set(handles.parameter1_name,'Enable','on');
    set(handles.parameter1_name,'Visible','on');
    set(handles.parameter2_name,'Enable','on');
    set(handles.parameter2_name,'Visible','on');
    set(handles.parameter3_name,'Visible','on');
    set(handles.parameter1_input,'Enable','on');
    set(handles.parameter1_input,'Visible','on');
    set(handles.parameter2_input,'Enable','on');
    set(handles.parameter2_input,'Visible','on');
    set(handles.parameter3_input,'Visible','on');
    set(handles.pushbutton_apply,'Enable','on');
    set(handles.pushbutton_apply,'Visible','on');
    set(handles.pushbutton_export,'Visible','on');
    set(handles.pushbutton_apply,'Visible','on');
    switch get(handles.parameter3_input,'Value');
        case 1          % plot
            if(length(currentprocessingarray.Originaldata)>0)
                set(handles.pushbutton_export,'Enable','on');
                set(handles.pushbutton_apply,'Enable','on');
            else
                set(handles.pushbutton_apply,'Enable','off');
                set(handles.pushbutton_export,'Enable','off');
            end
        case 3          %data reduction
            if(length(currentprocessingarray.reducedata)>0)
                set(handles.pushbutton_export,'Enable','on');
                set(handles.pushbutton_apply,'Enable','on');
            else
                set(handles.pushbutton_apply,'Enable','off');
                set(handles.pushbutton_export,'Enable','off');
```

```
            end
        case 2          %data denoise
            if(length(currentprocessingarray.denoisedata)>0)
                set(handles.pushbutton_export,'Enable','on');
                set(handles.pushbutton_apply,'Enable','on');
            else
                set(handles.pushbutton_apply,'Enable','off');
                set(handles.pushbutton_export,'Enable','off');
            end
    end
    set(handles.parameter1_name,'Style','text');
    set(handles.parameter1_name,'String','Fluid Type');
    set(handles.parameter1_input,'Style','popupmenu');
    set(handles.parameter1_input,'string','DD|Shutin_BU|Ratedrop_BU');
    switch currentprocessingarray.periodtype
        case 1
            set(handles.parameter1_input,'Value',1);
        case -1
            set(handles.parameter1_input,'Value',2);
        case -2
            set(handles.parameter1_input,'Value',3);
    end
    set(handles.parameter2_name,'String','Period Number');
    tempstring = '0';    sep= '|';    Numtotal = 0;
    for i=1:1:length(furtherprocessingarray)
        if(currentprocessingarray.periodtype…
          ==    furtherprocessingarray(i).periodtype)
            Numtotal = Numtotal + 1;
            if(Numtotal == 1)
                tempstring = num2str(Numtotal);
            else
                tempstring = [tempstring,sep,num2str(Numtotal)];
            end       end
    end
```

```
                set(handles.parameter2_input,'Style','popupmenu');
                set(handles.parameter2_input,'String',tempstring);
                if(currentprocessingarray.periodnum == 0)
                        set(handles.parameter2_input,'Value',1);
                else
               set(handles.parameter2_input,'Value',currentprocessingarray.periodnum);
                end
                set(handles.parameter3_name,'String','Operation');
                set(handles.parameter3_input,'Style','popupmenu');
          set(handles.parameter3_input,'String','Plot|Further Denoise|Data Reduction');
          end
          otherwise
    end
       if(state == 1)
            updatedhandles = handles;              % update handles
            varargout(1) =    {updatedhandles};
       else
            varargout(1) =    {updatedhandles};
       end
       return;
end
```

Wavelet function

```
% ------------------------------------------------------------------
function [a,d] = dwtremovaleps(x,varargin)
nbIn = nargin;
if nbIn < 2        error('Not enough input arguments.');
elseif nbIn > 7        error('Too many input arguments.');      end
if ischar(varargin{1})
      [Lo_D,Hi_D] = wfilters(varargin{1},'d'); next = 2;
else
      Lo_D = varargin{1}; Hi_D = varargin{2};    next = 3;
end
global DWT_Attribute
if isempty(DWT_Attribute)    DWT_Attribute = dwtmode('get'); end
```

```
dwtEXTM = DWT_Attribute.extMode; % Default: Extension.
shift      = DWT_Attribute.shift1D; % Default: Shift.
for k = next:2:nbIn-1
        switch varargin{k}
           case 'mode'    , dwtEXTM = varargin{k+1};
           case 'shift' , shift      = mod(varargin{k+1},2);
        end
end
lf = length(Lo_D); lx = length(x);
first = 2-shift; flagPer = isequal(dwtEXTM,'per');
if ~flagPer
        lenEXT = lf-1; last = lx+lf-1;
else
        lenEXT = lf/2; last = 2*ceil(lx/2);
end
y = wextend('1D',dwtEXTM,x,lenEXT);z = wconv1(y,Lo_D,'valid');
a = z(first:1:last);z = wconv1(y,Hi_D,'valid');
d = z(first:1:last);
```

Outlier removal function

```
% ------------------------------------------------------------------
%        this function is use to remove the outlier from original signal.
function varargout = outlierremovalfeps(x,varargin)
removlsignal = [];
varargout{1} = removlsignal;
nbIn = nargin;
if nbIn < 1
   error('Not enough input arguments.');
elseif nbIn > 7
   error('Too many input arguments.');
end
Originalsignal = x;
switch nargin
        case 2
```

```
            parameter = varargin{1};
    otherwise
end
waveletname = 'haar';
[cA,cD] = dwtremovaleps(Originalsignal,waveletname);
meanvalue = 0;      stdvalue = 0;           threshold =0;
maxvalueofcD = max(cD); minvalueofcD = min(cD);
if meanvalue == 0
    meanvalue = trimmean(cD,5);
elseif   meanvalue >= maxvalueofcD & meanvalue <= maxvalueofcD
    error('Wrong meanvalue of input!');
end
if stdvalue == 0
    stdvalue = iqr(cD);
elseif stdvalue >= maxvalueofcD & stdvalue <= maxvalueofcD
    error('Wrong standard deviation of input!');
end
if(threshold == 0)
    threshold = 20*stdvalue;
elseif threshold > abs(maxvalueofcD) & threshold > abs(minvalueofcD)
    error('Wrong threshold of input!');
end
grouparray = groupdataeps(cD,meanvalue,threshold,15);
outlierarray = selectoutliereps(grouparray);
for(i = 1:1:length(outlierarray))
    addflag = 1;      value = outlierarray(i).dvalue;
    m = size(value);
    if(value(1,1) == 1)
        for(j = value(m(1,1),1):-1:value(1,1))
            cD(j) = 0;       cA(j) = cA(j+addflag);      addflag = addflag + 1;
        end
    else
        for(j = value(1,1):1:value(m(1,1),1))
            cD(j) = 0;      cA(j) = cA(j-addflag);    addflag = addflag + 1;
```

```
            end     end
end
cA = cA(1:2:length(cA)); cD = cD(1:2:length(cD));
newsignal = idwt(cA,cD,waveletname);
varargout{1} = newsignal(1:length(newsignal));
return;

% ------------------------------------------------------------------
function flag = writestructeps(filename,pathname,dataset,varargin)
        nbIn = nargin;
    if(nargin == 4)   openstate = varargin{1};   end
    if(nargin == 3)   openstate = 'w';   end
    flag = 0;
    if isequal(filename,0) | isequal(pathname,0)
        disp('wrong filename and pathname');
    else
        flag = 1;
        if(isstruct(dataset))
            fid = fopen([pathname filename],openstate);
            if(fid ~= -1)
                names = fieldnames(dataset);
                for(i=1:1:length(names))
                    t = names{i};       temp = [];   temp = dataset.(t);
                    if(size(temp,2) == 1)
                        fprintf(fid,'%s:\n',names{i});
                    else
                        fprintf(fid,'%s:\n',names{i});
                        fprintf(fid,'%d\n',size(temp,1));
                        fprintf(fid,'%d\n',size(temp,2));
                    end
                    if(~iscellstr(temp) & ~ischar(temp)) %#ok<AND2>
                        % print the float data
                        for (k=1:1:size(temp,1))
                            for(j= 1:1:size(temp,2))
```

```
                                        if(isinteger(temp))
                                            fprintf(fid,'%12i',temp(k,j));
                                        else
                                            str = num2str(temp(k,j));
                                            fprintf(fid,'%s      ',str);
                                        end                 end
                                    fprintf(fid,'\n');              end
                                fprintf(fid,'\n');
                            else
                                fprintf(fid,'%s \n',temp);
                                fprintf(fid,' \n');
                            end         end         end
                        fclose(fid);
                end     end
return
```

Identification of BU and DD function

```
% --------------------------------------------------------------------
function [BUDD] = distinguishBUDD_functioneps(x,originalm,filetypedata,varargin)
account = x(:,1); time = x(:,2); pressure = x(:,3); xi = time;
yi = pressure;
switch nargin
    case 3
        minthrethord = 0.8;   interval = 0.1;   samplenumber = 10;   filetypedata = 1;
    case 6
      minthrethord =varargin{1};interval = varargin{2};samplenumber = varargin{3};
    otherwise
end
[a,d] = dwtremovaleps(yi,'haar');   lengthtemp = size(d);
mindata = minthrethord;   maxdata   = 500;     account = 0;
impulse = [0.00 0.00];
for i = 1:1:lengthtemp(1)
    if( abs(d(i)) <= maxdata & abs(d(i)) >= mindata)
        account = account + 1; impulse(account,1) = xi(i);
        impulse(account,2) = d(i);
```

```
    else
        d(i) = 0;
    end;
end
sigimp = [];    tempsigimp = [];    flag = 0; group = 0; account = size(impulse);
accountnum = account(1); group = 1; groupsize = 1; Time = impulse(1,1);
amplitude = impulse(1,2);
if( amplitude > 0)    flag = 1; else flag = -1; end
tempsigimp = [group,groupsize,flag,Time,amplitude];
for i=2:1:account(1)
    tempinterval =    impulse(i,1) - impulse(i-1,1);
    if (abs(tempinterval) < interval)
        groupsize = groupsize + 1;      Time = impulse(i,1);
        amplitude = impulse(i,2);
        if( amplitude > 0)      flag = flag + 1;              else
          flag = flag - 1;              end
        tempmatrix = [group,groupsize,flag,Time,amplitude];
        tempsigimp = [tempsigimp;tempmatrix];
    else
        tempsig = size(tempsigimp);
        if(tempsig(1) > samplenumber)
                sigimp = [sigimp;tempsigimp];      group = group +1;
                tempsigimp = [];      groupsize = 1;
            Time = impulse(i,1);          amplitude = impulse(i,2);
            if( amplitude > 0)      flag =    1;         else
               flag =    - 1;         end
            tempsigimp = [group,groupsize,flag,Time,amplitude];
        else
            groupsize = 1; tempsigimp = [];
            Time = impulse(i,1);    amplitude = impulse(i,2);
            if( amplitude > 0)        flag =    1;
            else      flag = - 1;         end
            tempsigimp = [group,groupsize,flag,Time,amplitude];
        end
```

```
    end
    tempsig = size(tempsigimp);
    if(i == account(1) & tempsig(1) > samplenumber)
        sigimp = [sigimp;tempsigimp];
    end
end
if(size(sigimp,1) == 0)    BUDD = [];    return;    end
BeginPressure = time(1); temp = size(time); EndPressure = time(temp(1));

BUDD = [];TEMPBUDD = [];NUMBER = 0; BEGIN = 0;
END = 0; VALUE = 0; group = 0; INTERVAL = 0;
if(sigimp(1,4) > BeginPressure + interval)
    NUMBER = sigimp(1,3);
    if(NUMBER > 0)
        NUMBER = -1;          VALUE = 0;
    else
        NUMBER = 1;    VALUE = sigimp(1,5);
    end
    END =    sigimp(1,4);    INTERVAL = END - 0;
    BUDD = [group,NUMBER,0,END,INTERVAL,VALUE];
    NUMBER = sigimp(1,3); BEGIN = sigimp(1,4);
    group = sigimp(1,1); VALUE = sigimp(1,5); END = BEGIN;
    TEMPBUDD = [group,NUMBER,BEGIN,END,INTERVAL,VALUE];
else
group = sigimp(1,1); NUMBER = sigimp(1,3); BEGIN = sigimp(1,4); END = BEGIN;
end
account = size(sigimp);
for i = 2:1:account(1)
    if(group == sigimp(i,1))
        if(abs(sigimp(i,5)) > abs(VALUE))    VALUE = sigimp(i,5); end;
        TEMPBUDD(1,6) = VALUE;
    else
        TEMPBUDD(1,4) = sigimp(i,4);
        TEMPBUDD(1,5) = TEMPBUDD(1,4) - TEMPBUDD(1,3);
```

```
            BUDD = [BUDD;TEMPBUDD];
            group = group + 1;
            if(abs(sigimp(i,3)) == 1)
                NUMBER = sigimp(i,3); BEGIN = sigimp(i,4);
                END = BEGIN;
            end;
            VALUE = sigimp(i,5);
           TEMPBUDD = [group,NUMBER,BEGIN,END,VALUE];
        end
        if(i == account(1))
            TEMPBUDD(1,4) = EndPressure;
            TEMPBUDD(1,5) = TEMPBUDD(1,4) - TEMPBUDD(1,3);
            TEMPBUDD(1,6) = sigimp(i,5);
            BUDD = [BUDD;TEMPBUDD];
        end
end
originalt = originalm(:,2); originalp = originalm(:,3);
switch filetypedata
        case 1          widowsize = 15;
        case 2          widowsize = 5;
        otherwise
end
for (i=1:1:size(BUDD,1)-1)
        breakpoint = find(originalt>=BUDD(i,4),1);
        flag = 0; temparray = [];temparrayt = [];movepoint = 0;
        rightpoint = 0;
        while(flag < 2 &    breakpoint+movepoint < size(originalt,1))
        temparrayt = originalt(breakpoint-widowsize+movepoint:breakpoint+movepoint);
        temparray = originalp(breakpoint-widowsize+movepoint:breakpoint+movepoint);
        mu = trimmean(temparray,10); sigma = iqr(temparray);
        if(abs(originalp(breakpoint+movepoint)- mu) > 3*sigma )
            flag = flag +1;
            if(flag ==1) rightpoint = breakpoint+movepoint-1; end
        end    movepoint = movepoint+1;        end
```

```
    if(flag < 2)        rightpoint = breakpoint; end
    BUDD(i,4) = originalt(rightpoint); BUDD(i,5) = BUDD(i,4) - BUDD(i,3);
    BUDD(i+1,3) = originalt(rightpoint);
end
tempBUDD = [];
for(i=1:1:size(BUDD,1))
    if(BUDD(i,5) > 0)     tempBUDD = [tempBUDD ; BUDD(i,:)]; end
end
BUDD = tempBUDD; beginpressure = 0; endpressure = 0;
for i=1:1:size(BUDD,1)
    beginindex = find(originalt>=BUDD(i,3),1);
    beginpressure = originalp(beginindex);
    endindex = find(originalt>=BUDD(i,4),1);
    endpressure = originalp(endindex);
    totalpercent =0; countflag = 0;
    if(endindex>beginindex+1)
        for(j =beginindex+1:1:endindex)
            if(originalp(j)>originalp(beginindex)) countflag = countflag + 1; end
        end
        if(abs(countflag*100/(endindex-beginindex)) > 80)
            BUDD(i,2) = -1;     else
            BUDD(i,2) = 1;    end
    else
        BUDD = [];     return;
    end         end
for i=1:1:size(BUDD,1)-1
    if(BUDD(i,2) == -1 && BUDD(i+1,2) == -1) BUDD(i,2) = -2; end
end
return ;
% ------------------------------------------------------------------
%         Groupdata function classify the original data into different group
function grouparray = groupdataeps(x,varargin)
detailsignal = [];grouparray = [];
groupstruct = struct('groupnumber',[],'dvalue',[]); tempstruct = groupstruct;
```

```
nbIn = nargin;
if nbIn < 3    error('Not enough input arguments.');
elseif nbIn > 7 error('Too many input arguments.'); end
detailsignal = x;
switch nargin
      case 4
            meanvalue = varargin{1};      threshold = varargin{2};
            numberdata = varargin{3};      otherwise
end
groupnumber = 0; flagofgroup = 1; temp = [];
temparray = [];flagofnumber = 0;
while flagofnumber == 0
for i =1:1:length(detailsignal)
      temp = abs(detailsignal(i) - meanvalue)-threshold;
      if(temp>0)
            temp = [i detailsignal(i)];    temparray = [temparray ;temp]; temp = [];
      else
            if length(temparray) > 0 & length(temparray)<= numberdata
                  groupnumber = groupnumber + 1;
                  tempstruct.groupnumber = groupnumber;
                  tempstruct.dvalue = temparray;
                  grouparray = [grouparray tempstruct];    temparray = [];
                  flagofnumber = 1;
            elseif length(temparray) > numberdata
                  flagofnumber = 0;      threshold = threshold*1.1;
                  temp = [];      temparray = [];    grouparray= [];break;
            end           end      end
end
flag = 1; groupold = grouparray(1);temparray = grouparray(1);
for(i=2:1:length(grouparray))
      groupnew = grouparray(i);        m = size(groupold.dvalue);
      if(groupnew.dvalue(1,1)-groupold.dvalue(m(1,1),1) >=3)
            flag = flag + 1;              groupnew.groupnumber = flag;
            temparray = [temparray groupnew];              groupold = groupnew;
```

```
    else
        temparray(flag).dvalue = [temparray(flag).dvalue ; groupnew.dvalue];
        groupold = temparray(flag);
    end     end
grouparray = temparray;
return;
```

Data compression function

```
% ------------------------------------------------------------------
%       This function is used to compress signal
function compressdata = compressfeps(A)
compressdata = [];rate = 10;time = A(:,1);x = A(:,2);compressionrate = 10;
lx = 0;lx = length(x);numtotalstep = floor(lx/10);numtimestep = floor(numtotalstep/2);
numdetailstep = numtotalstep - numtimestep;
[Lo_D,Hi_D] = wfilters('haar','d');
lf = length(Lo_D);   shift = 0;   first = 2 - shift;
dwtEXTM = 'sym';   flagPer = isequal(dwtEXTM,'per');
if ~flagPer
    lenEXT = lf-1; last = lx+lf-1;
else
    lenEXT = lf/2; last = 2*ceil(lx/2);
end
y = wextend('1D',dwtEXTM,x,lenEXT);
z = wconv1(y,Hi_D,'valid');d = z(first:1:last);
maxvalue = max(abs(d));minvalue = 0;
mindetailstep = floor(numdetailstep*0.85);stepadd = 0;
numstep = 0;timestep =0;flag=-1;
shreshold = (maxvalue + minvalue)/2+stepadd;
while flag ~= 0
    timestep = timestep + 1;
    if(timestep > 600) flag = 0; else numstep = 0;
    for s = 1:1:lx
        if(abs(d(s)) >= shreshold)          numstep = numstep+1;          end
    end
```

```
    if (numstep > numdetailstep)
        flag = 1;           shreshold = shreshold * 1.02;
    elseif (numstep < mindetailstep)
        flag = -1;          shreshold = shreshold * 0.98;
    else
        flag = 0;
    end     end
end
flag = 0;     numdetailstep = numstep;temp = 1;selectTime = [];selectPressure = [];
for s = 1:1:lx
     if(temp == s)
         selectTime = [selectTime time(s)];   selectPressure = [selectPressure x(s)];
         temp = temp + compressionrate*2;
     else
        if(abs(d(s)) >= shreshold)
         selectTime = [selectTime time(s)];   selectPressure = [selectPressure x(s)];
        end       end
end
compressdata = [selectTime' selectPressure'];
return;
```

Tranditional well testing module

Well testing function

```
% ------------------------------------------------------------------
%%      This file is used   to analysis the BU and DD information
function welltesting(varargin)
for k = 1:1:length(varargin)
    switch k
        case 1          handles = varargin{k};
        case 2          ptr = varargin{k};
        case 3          rate = varargin{k};
        case 4          fluidestate = varargin{k};
        case 5          analysis_type = varargin{k};
        otherwise
```

```
                disp([varargin{k} ' is not a right input element!']);
        end end
Pi = 4828;    temp = size(ptr);sizeofm = temp(2);rowsize = temp(1);
oldtime = ptr(1,1);newtime = ptr(1,1);ptr_reduce = ptr(1,:);
for(i = 2:1:rowsize)
        newtime = ptr(i,1);
        if(newtime ~= oldtime)        ptr_reduce = [ptr_reduce ; ptr(i,:)];        end
        oldtime = newtime;
end
if(fluidestate == 2)
        tp = ptr_reduce(1,1);      ptr_reduce(:,1) = ptr_reduce(:,1) - ptr_reduce(1,1);
        for(i=2:1:sizeofm) DeltP_reduce(:,i-1) = ptr_reduce(:,i) - ptr_reduce(1,i); end
        Time = (ptr_reduce(:,1).*tp./(tp + ptr_reduce(:,1)));
end
if(fluidestate == 1)
        ptr_reduce(:,1) = ptr_reduce(:,1) - ptr_reduce(1,1);
        for(i=2:1:sizeofm)    DeltP_reduce(:,i-1) = Pi - ptr_reduce(:,i); end
        Time = ptr_reduce(:,1);
end
set(handles.axes_black_box_model,'Visible','off');
set(get(handles.axes_black_box_model,'Children'),'Visible','off');
set(handles.ax2_black_box_model,'Visible','off');
set(get(handles.ax2_black_box_model,'Children'),'Visible','off');
        switch    analysis_type
        case 1
        axes(handles.axes_black_box_model); ax1 = gca;
        set(handles.axes_black_box_model,'Visible','on');        cla;
        for(i=2:1:sizeofm)
                plot(ptr_reduce(:,1),ptr_reduce(:,i),'.-','Color'); hold on;
        end
        xlabel('Time(hours)','FontSize',14);      ylabel('Pressure(psi)','FontSize',14);
        title('Cartesian plot','FontSize',16,'FontWeight','bold');        legend('show');
        hold off;
        case 2
```

```
    axes(handles.axes_black_box_model);      ax1 = gca;
    set(handles.axes_black_box_model,'Visible','on');      cla;
    for(j=1:1:sizeofm-1)
    loglog(Time,DeltP_reduce(:,j),'o','Color',[1*(j/(sizeofm-1)) 0 1-(j/(sizeofm-1))]);
        hold on;
        lengthdeltp = [];          lengthlogTime = [];          lengthresult = [];
        logTime = log(Time);          lengthdeltp = length(logTime);
        for(i = lengthdeltp :-1:2)
            lengthdeltp(i) = DeltP_reduce(i,j) - DeltP_reduce(i-1,j);
            lengthlogTime(i) = logTime(i) - logTime(i-1);
            lengthresult(i) = lengthdeltp(i)/lengthlogTime(i);
        end
       loglog(Time(3:length(logTime)),lengthresult(3:length(lengthresult)),…
       'o','Color',[1*((j)/(sizeofm-1))   0   1-((j)/(sizeofm-1))]);          hold   on;
   end
    xlabel('Equivalent Time(hours)','FontSize',14);
    ylabel('DeltaP - Pressure(psi)','FontSize',14);
    title('Log-Log Plot','FontSize',16,'FontWeight','bold');      hold off;
    case 3
    axes(handles.axes_black_box_model);      ax1 = gca;
    set(handles.axes_black_box_model,'Visible','on');      cla;
    for(i=1:1:sizeofm-1)
   semilogx(Time,DeltP_reduce(:,i),'.','Color',[1*(i/(sizeofm-1)) 0 1-(i/(sizeofm-1))]);
        hold on;
    end
    xlabel('Equivalent Time(hours)','FontSize',14);
    ylabel('DeltaP - Pressure(psi)','FontSize',14);
    title('Radial Flow Plot','FontSize',16,'FontWeight','bold');
    hold off;          otherwise
end
return;
```

initial the interface function

```
% --------------------------------------------------------------------
```

```
%          This function is used to initial the interface with the input
function   InitialanalysisInterface(handles)
           set(handles.axes_well_plot,'Visible','off');
           set(get(handles.axes_well_plot,'Children'),'Visible','off');
           set(handles.axes2_well_plot,'Visible','off');
           set(get(handles.axes2_well_plot,'Children'),'Visible','off');
           set(handles.ax2_black_box_model,'Visible','off');
           set(get(handles.ax2_black_box_model,'Children'),'Visible','off');
           set(handles.axes_black_box_model,'Visible','off');
           set(get(handles.axes_black_box_model,'Children'),'Visible','off');
           h = findobj('Tag','uipanel_bottom_state_well');       set(h,'Visible','off');
           hchildren = get(h,'Children');
           for(i=1:1:length(hchildren))    set(hchildren(i),'Visible','off');       end
           h = findobj('Tag','uipanel_left_control_well'); set(h,'Visible','off');
           h = findobj('Tag','uipanel_black_box_model'); set(h,'Visible','off');
           hchildren = get(h,'Children');
           for(i=1:1:length(hchildren)) set(hchildren(i),'Visible','off'); end
           h = findobj('Tag','uipanel_welltest_analysis');     set(h,'Visible','on');
           h = findobj('Tag','uipanel_select_BU_DD');         set(h,'Visible','on');
           hchildren = get(h,'Children');
           for(i=1:1:length(hchildren))
                 set(hchildren(i),'Visible','on'); set(hchildren(i),'Enable','off');
           end
           h = findobj('Tag','uipanel_denoise'); set(h,'Visible','on');
           hchildren = get(h,'Children');
           for(i=1:1:length(hchildren))
                 set(hchildren(i),'Visible','on'); set(hchildren(i),'Enable','off');
           end
           h = findobj('Tag','uipanel_welltesting_plot'); set(h,'Visible','on');
           hchildren = get(h,'Children');
           for(i=1:1:length(hchildren))
                 set(hchildren(i),'Visible','on'); set(hchildren(i),'Enable','off');
           end
           h = findobj('Tag','uipanel_compression'); set(h,'Visible','on');
```

```
hchildren = get(h,'Children');
for(i=1:1:length(hchildren))
    set(hchildren(i),'Visible','on'); set(hchildren(i),'Enable','off');
end
currentwell = handles.CurrentWell;              sep = '|';
WellNumber = length(handles.WellArray);
CurrentWellName = get(currentwell,'WellName'); stringlist = [];
flag = 1;
if(WellNumber ~= 0)
    for i=1:1:WellNumber-1
            string_wellname = get(handles.WellArray{i},'WellName');
            if(strcmp(string_wellname,CurrentWellName))
                flag = i;
            end
            stringlist = [stringlist,string_wellname,sep];
    end
    string_wellname = get(handles.WellArray{WellNumber},'WellName');
    if(strcmp(string_wellname,CurrentWellName))
        flag = WellNumber;
    end
    stringlist = [stringlist,string_wellname];
    set(handles.popupmenu_analysis_wellname,'String',stringlist);
    set(handles.popupmenu_analysis_wellname,'Value',flag);
    set(handles.text_analysis_wellname,'Enable','on');
    set(handles.popupmenu_analysis_wellname,'Enable','on');
else
    stringlist = ['None',sep]
    set(handles.popupmenu_analysis_wellname,'String',stringlist);
    set(handles.popupmenu_analysis_wellname,'Value',1);
    set(handles.text_analysis_wellname,'Enable','on');
    set(handles.popupmenu_analysis_wellname,'Enable','on');
end
pdgpressure = length(currentwell.WellPDGPressure);
recoverrate = currentwell.WellProduction;
```

```
recoverrate = recoverrate{4};
if(length(pdgpressure) > 0 & length(recoverrate) > 0)
    set(handles.popupmenu_select_type,'Enable','on');
    set(handles.text_select_type,'Enable','on');
    type = get(handles.popupmenu_select_type,'Value');
    switch type
        case 1    %% find DD
            totalDD = 0;        newrate = 0;        oldrate = 0;
            for i=1:1:length(recoverrate)-1
                newrate = recoverrate(i,2);
                if(newrate > oldrate)
                    totalDD = totalDD + 1;
                end
                oldrate = newrate;
            end
        case 2    %%Shutin_BU
            totalDD = 0;              newrate = 0;
            for i=1:1:length(recoverrate)-1
                newrate = recoverrate(i,2);
                if(newrate == 0 )
                    totalDD = totalDD + 1;
                end
            end
        case 3    %%Ratedrop_BU
            totalDD = 0; newrate = 0; oldrate = 0;
            for i=1:1:length(recoverrate)-1
                newrate = recoverrate(i,2);
                if(newrate < oldrate & newrate ~= 0)
                    totalDD = totalDD + 1;
                end
                oldrate = newrate;
            end
        otherwise
    end
```

```
                number = get(handles.popupmenu_shutin_Number,'Value');
                sep = '|'; stringlist = [];flag = 1;
                if(totalDD ~= 0)
                    for i=1:1:totalDD-1 stringlist = [stringlist,int2str(i),sep]; end
                        stringlist = [stringlist,int2str(totalDD),sep,'all'];
                        set(handles.popupmenu_shutin_Number,'String',stringlist);
                        set(handles.popupmenu_shutin_Number,'Value',1);
                        set(handles.popupmenu_shutin_Number,'Enable','on');
                        set(handles.text_select_number,'Enable','on');
                        set(handles.pushbutton_plot_ddbu,'Enable','on');
                else
                        stringlist = ['None',sep]
                        set(handles.popupmenu_shutin_Number,'String',stringlist);
                        set(handles.popupmenu_shutin_Number,'Value',1);
                        set(handles.popupmenu_shutin_Number,'Enable','on');
                        set(handles.text_select_number,'Enable','on');
                end
            else
                set(handles.popupmenu_select_type,'Value',1);
                stringlist = ['None',sep]
                set(handles.popupmenu_shutin_Number,'String',stringlist);
                set(handles.popupmenu_shutin_Number,'Value',1);
            end
        if(strcmp(get(handles.pushbutton_plot_ddbu,'Enable'),'on'))
            h = findobj('Tag','uipanel_denoise'); set(h,'Visible','on');
            hchildren = get(h,'Children');
            for(i=1:1:length(hchildren)) set(hchildren(i),'Enable','on'); end
        end
end
```

Numerical well testing model

```
% ------------------------------------------------------------------
function varargout = numericalwelltesting(varargin)
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',              mfilename, ...
                   'gui_Singleton',    gui_Singleton, ...
                   'gui_OpeningFcn', @forwardmodelling_OpeningFcn, ...
                   'gui_OutputFcn',    @forwardmodelling_OutputFcn, ...
                   'gui_LayoutFcn',    [] , ...
                   'gui_Callback',      []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% ------------------------------------------------------------------
function forwardmodelling_OpeningFcn(hObject, eventdata, handles, varargin)

Case= struct('Name','None','comment','None','firsttime','None',…
'lasttime','None','group',[]);
handles.currentcase = Case();handles.casearray = [];
handles.currentperiod = [];handles.groupvalueout = [];
currentexperiment =    experimentclass ();
handles.currentexperiment = currentexperiment;
handles.experimentarray = {};
set(handles.tag_uipanel_exeperiment,'Visible','off');
set(get(handles.tag_uipanel_exeperiment,'Children'),'Visible','off');
set(handles.tag_menu_forward_modeling,'Enable','off');
if(nargin>0)
    switch nargin
        case 4
            handles.currentperiod    = varargin{1};
        otherwise
    end
end
```

```
guidata(hObject, handles);
if(length(handles.currentperiod)>0)
      set(handles.tag_menu_forward_modeling,'Enable','on');
end
handles.output = hObject;          guidata(hObject, handles);
uiwait(handles.tag_figure_forwardmodeling);
% --------------------------------------------------------------------
% --- Outputs from this function are returned to the command line.
function varargout = forwardmodelling_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
delete(handles.tag_figure_forwardmodeling);
% --------------------------------------------------------------------
function tag_menu_new_project_Callback(hObject, eventdata, handles)
set(handles.tag_menu_forward_modeling,'Enable','on');
[FileName,PathName,FilterIndex] = uiputfile({'*.fm','Forward Modeling file(*.fm)';...
      '*.*','All file(*.*)'});
if isequal(FileName,0) | isequal(PathName,0)
    disp('User selected Cancel');
else
    disp(['User selected',fullfile(PathName,FileName)]);
    switch FilterIndex
     case 1
        handles.FileName = FileName;           handles.PathName = PathName;
        handles.FilterIndex = FilterIndex;    disp('create a new forward modeling file!');
        fid = fopen([PathName,FileName],'w');
        fprintf(fid,'************************************************\n');
        fprintf(fid,'*\n');
        fprintf(fid,'*       This file is used to contain the forward modeling data. \n');
        fprintf(fid,'*\n');
        fprintf(fid,'*       Date: ');     fprintf(fid,'%s\n',date);
        fprintf(fid,'*\n');
        fprintf(fid,'************************************************\n\n');
        status = fclose(fid);
     otherwise     disp('Unknown method.'); end
```

```
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_menu_load_project_Callback(hObject, eventdata, handles)
set(handles.tag_menu_forward_modeling,'Enable','off');
set(handles.tag_uipanel_exeperiment,'Visible','off');
set(get(handles.tag_uipanel_exeperiment,'Children'),'Visible','off');
set(get(handles.tag_uipanel_experiment_info,'Children'),'Enable','off');
set(get(handles.tag_uipanel_load_experiment_variable,'Children'),'Enable','off');
set(get(handles.tag_uipanel_design_experiment_case,'Children'),'Enable','off');
set(handles.tag_togglebutton_experiment_next,'Enable','off');

[FileName,PathName,FilterIndex] = uigetfile({'*.fm','Forward Modeling file(*.fm)';...
      '*.*','All file(*.*)'});
if isequal(FileName,0) | isequal(PathName,0)
    disp('User selected Cancel');
else
    disp(['User selected',fullfile(PathName,FileName)]);
    switch FilterIndex
     case 1
        handles.FileName = FileName;
        handles.PathName = PathName;
        handles.FilterIndex = FilterIndex;
        fid = fopen([handles.PathName,handles.FileName],'r');
        for(i = 1:1:9)    tline = fgetl(fid);      end
        experimentnumber = 0; experimentnumber = fscanf(fid,'%d');
        for(i=1:1:experimentnumber)
           handles.currentexperiment = loadexperiment(handles.currentexperiment,fid);
           handles.experimentarray{i} = handles.currentexperiment;
           if(i ~= experimentnumber)
              tline = fgetl(fid);    tline = fgetl(fid);
           end
        end
        status = fclose(fid);
```

```
        set(handles.tag_menu_forward_modeling,'Enable','on');
EclipseFilepathname = get(handles.currentexperiment,'OriginalEclipseFilepathname');
if(~logical(strcmp(EclipseFilepathname,'None')) & ~isequal(EclipseFilepathname,0))
                Detectvalue = eclread(EclipseFilepathname);
                if(isa(Detectvalue,'cell'))
                    handles.currentexperiment = set(handles.currentexperiment, ...
                        'VariableInformation',Detectvalue{1});
                    ChangedParameterArray = findpara(Detectvalue);
                    handles.currentexperiment = set(handles.currentexperiment, ...
                        'ChangedParameterArray',ChangedParameterArray);
                    newdata = UpdateDataInfo(handles,'save');
                    if(newdata{1} == 1)
                        handles.experimentarray = newdata{2};
                        handles.currentexperiment = newdata{3};
                    end       end         end
    otherwise          disp('Unknown method.');       end
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_menu_save_project_Callback(hObject, eventdata, handles)
savefmdata(hObject, eventdata, handles);
% --------------------------------------------------------------------
function tag_menu_save_as_Callback(hObject, eventdata, handles)
  [handles.FileName,handles.PathName,handles.FilterIndex] =…
  uiputfile('Forward Modeling file(*.fm)','Save Workspace As');
if isequal(handles.FileName,0) | isequal(handles.PathName,0)
    msgbox('User selected Cancel!','Save As Warning!!','warn');
    disp('User selected Cancel')
else
    savefmdata(hObject, eventdata, handles);
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_menu_exit_Callback(hObject, eventdata, handles)
```

```
user_response = exitmodaldlg('Title','Confirm Close');

switch user_response
case {'No'}
    delete(handles.tag_figure_forwardmodeling);      clear all;
case 'Yes'
    savefmdata(hObject, eventdata, handles);
    delete(handles.tag_figure_forwardmodeling);
end
% --------------------------------------------------------------------
function Tag_numerical_experiment_Callback(hObject, eventdata, handles)
IniNumExeInterface(handles);

% --------------------------------------------------------------------
function tag_pushbutton_add_experiment_Callback(hObject, eventdata, handles)
output = figexperimentinfo();
if(isa(output,'experimentclass'))
    handles.currentexperiment = output;
    newdata = UpdateDataInfo(handles,'save');
    if(newdata{1} == 1)
        handles.experimentarray = newdata{2};
        handles.currentexperiment = newdata{3};
    end
end
IniNumExeInterface(handles);    guidata(hObject, handles);

% --------------------------------------------------------------------
function tag_pushbutton_del_experiment_Callback(hObject, eventdata, handles)
string_experimentname = get(handles.currentexperiment,'ExperimentName');
if(~logical(strcmp(string_experimentname,'None')))
    %delet data
    newdata = UpdateDataInfo(handles,'Del');
    if(newdata{1} == 1)
     handles.experimentarray = newdata{2};handles.currentexperiment = newdata{3};
```

```
    end
    IniNumExeInterface(handles);
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_pushbutton_display_experiment_Callback(hObject, eventdata, handles)
output = figexperimentinfo('experimentinfo',handles.currentexperiment);
if(isa(output,'experimentclass'))
      string_experimentname = get(output,'ExperimentName');
      if(~logical(strcmp(string_experimentname,'None')))
          newdata = UpdateDataInfo(handles,'Del');
          if(newdata{1} == 1)     handles.experimentarray = newdata{2};    end
          handles.currentexperiment = output;
          newdata = UpdateDataInfo(handles,'save');
          if(newdata{1} == 1)
                handles.experimentarray = newdata{2};
                handles.currentexperiment = newdata{3};
          end
          IniNumExeInterface(handles);
      end
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_popupmenu_experiment_name_Callback(hObject, eventdata, handles)
namestr= get(handles.tag_popupmenu_experiment_name,'String');
currentname = namestr(get(handles.tag_popupmenu_experiment_name,'Value'),:);
if(~logical(strcmp(currentname,'None')))
    newdata = UpdateDataInfo(handles,'ChangeCurrentExp',currentname);
    if(newdata{1} == 1)    handles.currentexperiment = newdata{3};    end
    handles.casearray = get(handles.currentexperiment,'Case');
    numbercase = length(handles.casearray);
    if(numbercase == 0)
          Case = struct('Name','None','comment','None',....
                'firsttime','None','lasttime','None','group',[]);
```

```
            handles.currentcase = Case();
    else
            handeles.currentcase = handles.casearray{1};
    end
    IniNumExeInterface(handles);
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_pushbutton_load_eclipse_file_Callback(hObject, eventdata, handles)
[FileName,PathName,FilterIndex] = uigetfile({'*.DATA','Eclipse Data File(*.DATA)';..
     '*.*','All file(*.*)'});
if isequal(FileName,0) | isequal(PathName,0)
    disp('User selected Cancel');
else
    disp(['User selected',fullfile(PathName,FileName)]);
    handles.currentexperiment = set(handles.currentexperiment, ...
        'OriginalEclipseFilepathname',fullfile(PathName,FileName));
    Detectvalue = eclread(FileName,PathName);
    if(isa(Detectvalue,'cell'))
        handles.currentexperiment = set(handles.currentexperiment, ...
            'VariableInformation',Detectvalue{1});
        ChangedParameterArray = findpara(Detectvalue);
        handles.currentexperiment = set(handles.currentexperiment, ...
            'ChangedParameterArray',ChangedParameterArray);
        newdata = UpdateDataInfo(handles,'save');
        if(newdata{1} == 1)
            handles.experimentarray = newdata{2};
            handles.currentexperiment = newdata{3};
        end
    end
    IniNumExeInterface(handles);
end
guidata(hObject, handles);
% --------------------------------------------------------------------
```

```
function tag_togglebutton_view_variable_Callback(hObject, eventdata, handles)
EclipseFilepathname = get(handles.currentexperiment,'OriginalEclipseFilepathname');
if(~logical(strcmp(EclipseFilepathname,'None')) & ~isequal(EclipseFilepathname,0))
    Detectvalue = eclread(EclipseFilepathname);
    if(isa(Detectvalue,'cell'))
        handles.currentexperiment = set(handles.currentexperiment, ...
            'VariableInformation',Detectvalue{1});
        ChangedParameterArray = findpara(Detectvalue);
        handles.currentexperiment = set(handles.currentexperiment, ...
            'ChangedParameterArray',ChangedParameterArray);
        newdata = UpdateDataInfo(handles,'save');
        if(newdata{1} == 1)
    handles.experimentarray = newdata{2};handles.currentexperiment = newdata{3};
        end
  outputtemp = figkeywordvariable('keywordvariable',handles.currentexperiment);
    end
else
    msgbox('There is no Keywords to display!','Display Warning!!','warn');
end
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_pushbutton_add_Callback(hObject, eventdata, handles)
Case = struct('Name','None','comment','None','firsttime','None','lasttime',…
'None','group',[]);
fistcase = Case();output = figexperimentinfo('caseinfo',fistcase);
if(isstruct(output))
    if(~logical(strcmp(output.Name,'None')) & ~isequal(output.Name,0))
        handles.currentcase = output;
        handles.casearray = get(handles.currentexperiment,'Case');
        string_experimentname = output.Name;
        numberexperiment = length(handles.casearray);
        if(numberexperiment == 0)
            handles.casearray{1} = handles.currentcase;
        else
```

```
                for(i=1:1:numberexperiment)
if(logical(strcmp(string_experimentname,handles.casearray{i}.Name)))
                    handles.casearray{i} = handles.currentcase; break;
                else
                    if(i==numberexperiment)
                        handles.casearray{i+1} = handles.currentcase;
                    end       end      end end
handles.currentexperiment = set(handles.currentexperiment,'Case',handles.casearray);
            newdata = UpdateDataInfo(handles,'save');
            if(newdata{1} == 1)
                handles.experimentarray = newdata{2};
                handles.currentexperiment = newdata{3};
            end       end       end
IniNumExeInterface(handles);
guidata(hObject, handles);
% ---------------------------------------------------------------------
function tag_pushbutton_display_case_Callback(hObject, eventdata, handles)
casename = handles.currentcase.Name
if(~logical(strcmp(casename,'None')) & ~isequal(casename,0))
    % display currentcase
    output = figexperimentinfo('caseinfo',handles.currentcase);
end
IniNumExeInterface(handles);
guidata(hObject, handles);
% ---------------------------------------------------------------------
function tag_pushbutton_del_case_Callback(hObject, eventdata, handles)
newcasearray = [];    count = 1; numbercase = length(handles.casearray);
if(numbercase == 0)
    Case = struct('Name','None','comment','None',....
        'firsttime','None','lasttime','None','group',[]);
    handles.currentcase = Case();
else
    for i=1:1:numbercase
        string_tempname = handles.casearray{i}.Name;
```

```
            if(~logical(strcmp(handles.currentcase.Name,string_tempname)))
                newcasearray{count} = handles.casearray{i}
                count = count + 1;
            end
        end
        handles.casearray = newcasearray;
end
if(length(handles.casearray) == 0)
        Case = struct('Name','None','comment','None',....
            'firsttime','None','lasttime','None','group',[]);
        handles.currentcase = Case();
else
        handles.currentcase = handles.casearray{1};
end
handles.currentexperiment = set(handles.currentexperiment,'Case',handles.casearray);
newdata = UpdateDataInfo(handles,'save');
if(newdata{1} == 1)
        handles.experimentarray = newdata{2};
        handles.currentexperiment = newdata{3};
end
IniNumExeInterface(handles);
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_listbox_case_info_Callback(hObject, eventdata, handles)
strlist = get(handles.tag_listbox_case_info,'String');
currentvalue = get(handles.tag_listbox_case_info,'Value');
if(currentvalue ~= 1)
        currentname = strlist(currentvalue,:);
        currentname = strtrim(currentname);
        [token, rem] = strtok(currentname,' ');
        numbercase = length(handles.casearray);
        for i=1:1:numbercase
            string_tempname = handles.casearray{i}.Name;
            if(logical(strcmp(token,string_tempname)))
```

```
                handles.currentcase = handles.casearray{i};
                break;    end
        end
 handles.currentexperiment = set(handles.currentexperiment,'Case',handles.casearray);
        newdata = UpdateDataInfo(handles,'save');
        if(newdata{1} == 1)
            handles.experimentarray = newdata{2};
            handles.currentexperiment = newdata{3};
        end
end
IniNumExeInterface(handles);
guidata(hObject, handles);
% --------------------------------------------------------------------
function tag_pushbutton_operation_Callback(hObject, eventdata, handles)
experimentsetp = get(handles.tag_pushbutton_design_step,'Value');
switch experimentsetp
    case 1
        temparray = get(handles.currentexperiment,'ChangedParameterArray');
        if(length(temparray) > 0)
            groupvalue = figaddvalue(temparray,handles.currentcase.group);
            if(length(groupvalue) > 1)
                handles.currentcase.group = groupvalue{2};
            end
        else
            msgbox('There is no variable in this case!','Warning!!','warn');
        end
    case 2
        if(length(handles.currentcase.group) > 0)
            groupvalue = figgroupvalue(handles.currentcase.group);
            if(length(groupvalue) > 1)
                handles.currentcase.group = groupvalue{2};
            end
        else
            msgbox('There is no group in this case!','Warning!!','warn');
```

```
            end
        case 3
            if(length(handles.currentcase.group(1).Range)>0)
                tempgroup = handles.currentcase.group;
                groupinput{1,1} = tempgroup;
                   outexperiment = designexperiment(groupinput);
                temp = get(handles.currentexperiment,'OriginalEclipseFilepathname');
                [pathstr,name,ext,versn] = fileparts(temp);
                Original_PathName = [pathstr '\'];
                Original_PathName = strtrim(Original_PathName);
                Original_FileName = [name ext];
                Original_FileName = strtrim(Original_FileName);
                for i=1:1:length(outexperiment)
                    newfilename = [outexperiment(1,i).Name    '.DATA'];
                    newfilename = [handles.currentcase.Name newfilename];
                    ChangedParameterArray = …
                  get(handles.currentexperiment,'ChangedParameterArray');
          pathandname = writeeperiment(ChangedParameterArray,outexperiment(1,i),
                    Original_PathName,Original_FileName,newfilename);
              batchfilename{i} = [handles.currentcase.Name outexperiment(1,i).Name];
                end
           batchfilenamepath = [Original_PathName handles.currentcase.Name '.BAT'];
                fidw = fopen(batchfilenamepath ,'w');
                    for i=1:1:length(batchfilename)
                        Original_PathName = strtrim(Original_PathName);
                        teststr =    ['CALL $eclipse -data ' Original_PathName];
                        teststr = [teststr ['      -file        '] batchfilename{i}];
                        fprintf(fidw,'%s\n',teststr);
                    end
                status = fclose(fidw);
          user_response = exitmodaldlg('Title','Run Simulation','String','Do you want to
run now?');
                switch user_response
                case 'No'
```

```
                case 'Yes'
                        if(length(batchfilenamepath)>0)
                            str = batchfilenamepath;          dos(str);
                        end     end     end
    case 4
         if(length(handles.currentcase.group(1).Range)>0)
            temppathname = …
            get(handles.currentexperiment,'OriginalEclipseFilepathname');
            tempgroup = handles.currentcase.group;
            casename = handles.currentcase.Name;
            handles.groupvalueout = figanalysisexperiment ...
            (handles.currentperiod,tempgroup,temppathname,casename);
        else
            msgbox('There is no experiment for analysis!','Warning!!','warn');
        end
    otherwise
end
IniNumExeInterface(handles);
guidata(hObject, handles);
% ---------------------------------------------------------------------
function tag_togglebutton_experiment_next_Callback(hObject, eventdata, handles)
if length(handles.groupvalueout )>0
    handles.output = handles.groupvalueout;
    guidata(hObject, handles);
    uiresume(handles.tag_figure_forwardmodeling);
end
```

Read eclipse data file function

```
% ---------------------------------------------------------------------
%       this function is used to read ecl file for numerical experiment to
function outchange = eclread(varargin)
switch nargin
    case 1
        filepathname = varargin(1);             filepathname = filepathname{1};
```

```
            filepathname = strtrim(filepathname);
            [pathstr,name,ext,versn] = fileparts(filepathname);
            PathName = [pathstr '\'];              FileName = [name ext];
        case 2
            FileName = varargin(1);              PathName = varargin(2);
        otherwise
            outchange = 0;              return;
end
MarkArray = ['@' '#' '$'];
LineContainer = struct('OriginalLine','','Position',[]);
KeywordsContainer = struct('Range',[],'DetailLine',{});
ChangedContainer = struct('filetype',0,'Location',[],'DetailKeywords',{});
ChangedContainerArray = [];
NumberLineinkeywords = 0;
NumberKeywords = 0;
Numberinclude = 0;NumberOfLine = 0;filetype = 0;
ChangedContainer(1).filetype = filetype;
TempKeywordsContainerArray = {};TempContainerLineArray = {};
TempRange = [];flaginclude = 0; fid = fopen([char(PathName),char(FileName)],'r');
if fid ~= -1
    eofstat = feof(fid);            flagline = 0;
    while(eofstat == 0)
        NumberOfLine = NumberOfLine + 1;    templine = fgetl(fid);
        if(length(templine) ~= 0)
            FirstCharacter_Line = templine(1);
            if(flagline == 1 & FirstCharacter_Line ~= '#')
                FirstCharacter_Line = '@';
            end
            switch FirstCharacter_Line
                case '@'
                    flagline = 1;
                    NumberLineinkeywords = NumberLineinkeywords + 1;
                    if(length(templine) > 7)
                        if(templine(2:8)~='INCLUDE'& templine(2:8)~='include')
```

```
            flaginclude = 1;
        end
    else
        flaginclude = 1;
    end
    if(flaginclude)
        TempRange = [TempRange NumberOfLine];
        LineContainer.OriginalLine = templine;
        beginofvalue = 0;          endofvalue = 0;
        tempPosition = [];         flagposition = 0;
        numposition = 0;
        for(i=1:1:length(templine))
            if(templine(i) == '$')
                numposition = numposition + 1; beginofvalue = i;
                tempPosition(numposition,1) = beginofvalue;
                flagposition = 1;
            end
            if(flagposition == 1 & (i==length(templine) |
            int8(templine(i)) == 32 | int8(templine(i)) == 9 |
            int8(templine(i)) == 42 …
          | int8(templine(i)) == 47))
                flagposition = 0;
                if(endofvalue == 0)
                    if(i==length(templine))
                        endofvalue = i;
                    else
                        endofvalue = i-1;
                    end
                    tempPosition(numposition,2) = endofvalue;
                end
                endofvalue = 0;
            end
        end
        LineContainer.Position = tempPosition;
```

```
TempContainerLineArray{NumberLineinkeywords} = LineContainer;
    else
        TempRange = [TempRange NumberOfLine];
    end
case '#'    %-- '#' Mark the end of keywords
    flagline = 0;
  if(flaginclude == 1)
    NumberLineinkeywords = NumberLineinkeywords + 1;
    NumberKeywords = NumberKeywords + 1;
    TempRange = [TempRange NumberOfLine];
    LineContainer.OriginalLine = templine;
        beginofvalue = 0;    endofvalue = 0;
        tempPosition = [];   flagposition = 0;   numposition = 0;
        for(i=1:1:length(templine))
            if(templine(i) == '$')
                numposition = numposition + 1;
                beginofvalue = i;
                tempPosition(numposition,1) = beginofvalue;
                flagposition = 1;
            end
      if(flagposition == 1 & (i==length(templine) | int8(templine(i))
          …== 32 | int8(templine(i)) == 9 ))
                flagposition = 0;
                if(endofvalue == 0)
                    endofvalue = i;
                    tempPosition(numposition,2) = endofvalue;
                end
                endofvalue = 0;          end
        end
    LineContainer.Position = tempPosition;
empContainerLineArray{NumberLineinkeywords} = LineContainer;
    KeywordsContainer(1).Range = TempRange;
    KeywordsContainer.DetailLine = TempContainerLineArray;
TempKeywordsContainerArray{NumberKeywords} = KeywordsContainer;
```

```
                    TempKeywordsContainer = {};TempContainerLineArray = {};
                    NumberLineinkeywords = 0; TempRange = [];flaginclude = 0;
                else
                TempRange = [TempRange NumberOfLine];      beginwords = 0;
                endwords = 0;
                  for i=1:1:length(templine)
                      if(int8(templine(i)) == 39    & beginwords == 0)
                          beginwords = i;
                      end
                      if(int8(templine(i)) == 39    & beginwords ~= 0)
                          endwords = i;
                      end
                  end
                  FileName = templine(beginwords+1:endwords-1);
                  Path_Name = char(PathName);
                  includecontainer = eclread(FileName,Path_Name);
                  tempincludec = includecontainer{1,1};
                  tempincludec.filetype = 1;
                  tempincludec.Location = TempRange;
                 ChangedContainerArray = [ChangedContainerArray tempincludec];
                  TempKeywordsContainer = {};TempContainerLineArray = {};
                  NumberLineinkeywords = 0; TempRange = [];
                  end
               otherwise      flagline = 0; end
          end         eofstat = feof(fid);
        end
    status = fclose(fid);
    ChangedContainer.DetailKeywords = TempKeywordsContainerArray;
    ChangedContainerArray = [ChangedContainerArray ChangedContainer];
    outchange = {ChangedContainerArray};
else
    msgbox('Can not open the file!','Open File Warning!!','warn');
    outchange = 0;
end
```

```
return;
```

Find parameter function

```
% --------------------------------------------------------------------
function outpara = findpara(varargin)
switch nargin
    case 1
        inputvalue = varargin(1);
    otherwise
        outpara = 0; return;
end
Detectvalue = inputvalue{1,1};Detectvalue = Detectvalue{1,1};
ChangedParameter = struct('Number',0,'Loction',[],'FileLocation',[],'OriginalValue','');
ChangedParameterArray = [];NumberofP = 0;
for i = 1:1:length(Detectvalue)
    tempfilechanged = Detectvalue(1,i);      file_number = i;
    tempkeywordsc = tempfilechanged.DetailKeywords;
    for j = 1:1:length(tempfilechanged)
        tempkeywordsc = tempfilechanged.DetailKeywords;
        for k = 1:1:length(tempkeywordsc)
            templinechanged = tempkeywordsc{1,k};
            linenumbertotal = templinechanged.Range;
            templinechanged = templinechanged.DetailLine;
            for m = 1:1:length(templinechanged)
                linenumber = linenumbertotal(1,m);
                Tempvaluechanged = templinechanged(1,m);
                lineoriginalvalue = templinechanged(1,m)
                lineoriginalvalue = lineoriginalvalue{1}.OriginalLine;
                Tempvaluechangedtotal = Tempvaluechanged{1,1}.Position;
                if(length(Tempvaluechangedtotal) > 0)
                    sizelinevalue = size(Tempvaluechangedtotal);
                 for t = 1:1:sizelinevalue(1,1)
                    NumberofP = NumberofP + 1;
                    ChangedParameter.Number = NumberofP;
```

```
                    Tempvaluechanged = Tempvaluechangedtotal(t,:);
                    loction = [file_number linenumber Tempvaluechanged];
                    ChangedParameter.Loction = loction;
                    ChangedParameter.FileLocation = tempfilechanged.Location;
                    originalvalue = lineoriginalvalue(loction(1,3)+1:loction(1,4));
                    ChangedParameter.OriginalValue = originalvalue;
                    ChangedParameterArray = [ChangedParameterArray …
                ChangedParameter];
               end   end   end   end   end
end
outpara = ChangedParameterArray ;
return ;
% ------------------------------------------------------------------
%       this function is used to read ecl file for numerical experiment to
function outchange = loadeclfile(varargin)
switch nargin
    case 1          outchange = 0;          return;
   case 2          FileName = varargin(1);          PathName = varargin(2);
  otherwise        outchange = 0;          return;
end
wellptr = [];   time = 0; rate = 0; pressure = 0;
fid = fopen([char(PathName),char(FileName)],'r');
if fid ~= -1
    eofstat = feof(fid);
    for(i=1:1:6)          templine = fgetl(fid);      end
    while(eofstat == 0)
        a = fscanf(fid,'%f %f %f %f',[4 inf]);          templine = fgetl(fid);
        eofstat = feof(fid);
    end
else
  disp('Can not open the file!');      outchange = 0;      return;
end
a = a';wellptr = [a(:,1)*24 a(:,3) a(:,4)];wellptr = wellptr(1:length(wellptr),:);
outchange = wellptr;      return;
```

Welltest plot function

```
% ------------------------------------------------------------------
%%      This file is used    to analysis the BU and DD information
function welltestingf(varargin)
for k = 1:1:length(varargin)
    switch k
        case 1      ptr_reduce = varargin{k};
        case 2      fluidestate = varargin{k};
        case 3      analysis_type = varargin{k};
        case 4      figurecontrol = varargin{k};
        case 5      properties = varargin{k};    Pi    = properties{1,1};
        otherwise    disp([varargin{k} ' is not a right input element!']);
    end
end
temp = size(ptr_reduce);sizeofm = temp(2);rowsize = temp(1);
tp = ptr_reduce(1,1);
ptr_reduce(:,1) = ptr_reduce(:,1) - ptr_reduce(1,1);
if(fluidestate == 1)
    for(i=3:1:sizeofm) DeltP_reduce(:,i-2) = ptr_reduce(:,i) - ptr_reduce(1,i); end
    Time = (ptr_reduce(:,1).*tp./(tp + ptr_reduce(:,1)));
end
if(fluidestate == 2)
    for(i=3:1:sizeofm) DeltP_reduce(:,i-2) = Pi - ptr_reduce(:,i); end
    Time = ptr_reduce(:,1);
end
switch    analysis_type
    case 1
    if(figurecontrol(1))    figure; end
    for(i=3:1:sizeofm)
colorarray = [figurecontrol(3)/figurecontrol(4) 0 1-(figurecontrol(3)/figurecontrol(4))];
        plot(ptr_reduce(:,1),ptr_reduce(:,i),'.-','Color',colorarray);      hold on;
    end
    if(figurecontrol(2))
        xlabel('Time(hours)','FontSize',14); ylabel('P -- Pressure(psi)','FontSize',14);
```

```
        title('P vs Time    --- Cartesian plot','FontSize',16,'FontWeight','bold');
        legend('show');      hold off;
end
case 2
if(figurecontrol(1))      figure; end
for(j=1:1:sizeofm-2)
            colorarray = [figurecontrol(3)/figurecontrol(4) 0 …
            1-(figurecontrol(3)/figurecontrol(4))];
             loglog(Time,DeltP_reduce(:,j),'*','Color',colorarray);
        hold on; lengthdeltp = [];    lengthlogTime = [];
        lengthresult = [];    logTime = log(Time); lengthdeltp = length(logTime);
        for(i = lengthdeltp :-1:2)
              lengthdeltp(i) = DeltP_reduce(i,j) - DeltP_reduce(i-1,j);
              lengthlogTime(i) = logTime(i) - logTime(i-1);
              lengthresult(i) = lengthdeltp(i)/lengthlogTime(i);
        end
        colorarray = [figurecontrol(3)/figurecontrol(4) 0 …
            1-(figurecontrol(3)/figurecontrol(4))];
        loglog(Time(3:length(logTime)),lengthresult(3:length(lengthresult)),
       …'*','Color',colorarray);
        hold on;
end
if(figurecontrol(2))
        legend('show');        xlabel('Equivalent Time(hours)','FontSize',14);
        ylabel('DeltaP - Pressure(psi)','FontSize',14);
        title('Log-Log Plot','FontSize',16,'FontWeight','bold');        hold off;
end
case 3
if(figurecontrol(1)) figure; end
for(i=1:1:sizeofm-2)
      colorarray = [figurecontrol(3)/figurecontrol(4) 0…
        1-(figurecontrol(3)/figurecontrol(4))];
        semilogx(Time,DeltP_reduce(:,i),'.','Color',colorarray);
        hold on;
```

```
    end
    if(figurecontrol(2))
        switch fluidestate
            case 1      xlabel('Equivalent Time(hours)','FontSize',14);
            case 2    xlabel('Elapsed Time(hours)','FontSize',14);
            otherwise
        end
        ylabel('DeltaP - Pressure(psi)','FontSize',14);
        title('Radial Flow Plot','FontSize',16,'FontWeight','bold');
        hold off;
    end
    otherwise
end
```

Dynamic forward modeling modular

Main function

```
% ------------------------------------------------------------------
function varargout = lastloop(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',          mfilename, ...
                   'gui_Singleton',    gui_Singleton, ...
                   'gui_OpeningFcn', @lastloop_OpeningFcn, ...
                   'gui_OutputFcn',    @lastloop_OutputFcn, ...
                   'gui_LayoutFcn',    [] , ...
                   'gui_Callback',     []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
endif nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% ------------------------------------------------------------------
function lastloop_OpeningFcn(hObject, eventdata, handles, varargin)
handles.eventnum = -1;
```

```
guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;end
temp = UpdatelastloopInterface(handles);handles.output = hObject;
guidata(hObject, handles);
% --------------------------------------------------------------------
function varargout = lastloop_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% --------------------------------------------------------------------
function Menu_File_New_Callback(hObject, eventdata, handles)
handles.eventnum = 0;            guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1) handles = outhandles; end
guidata(hObject, handles);
if isequal(handles.FileName,0) | isequal(handles.PathName,0)
else
       temp = UpdatelastloopInterface(handles);
end
% --------------------------------------------------------------------
function pushbutton_all_loaddata_Callback(hObject, eventdata, handles)
handles.eventnum = 1;        guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)    handles = outhandles; end
guidata(hObject, handles);
temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function popupmenu_all_periodname_Callback(hObject, eventdata, handles)
temp =    handles.periodarray;
handles.currentperiod = temp(get(handles.popupmenu_all_periodname,'Value'));
guidata(hObject, handles);temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton_match_loop_show_period_Callback(hObject, eventdata, handles)
handles.eventnum = 2;            guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
```

```
if(state == 1)        handles = outhandles;end
guidata(hObject, handles);temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton_match_loop_run_Callback(hObject, eventdata, handles)
handles.eventnum = 3;          guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1) handles = outhandles; end
guidata(hObject, handles);
temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton_match_loop_compare_Callback(hObject, eventdata, handles)
handles.eventnum = 4;guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;end
guidata(hObject, handles);
temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton_match_loop_study_Callback(hObject, eventdata, handles)
handles.eventnum = 5;            guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;end
guidata(hObject, handles); temp = UpdatelastloopInterface(handles);

% --------------------------------------------------------------------
function pushbutton_match_loop_next_Callback(hObject, eventdata, handles)
handles.eventnum = 6;    guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;end
guidata(hObject, handles);temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton_match_loop_addplot_Callback(hObject, eventdata, handles)
handles.eventnum = 7;          guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;end
```

```
guidata(hObject, handles);temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton1_match_loop_yes_Callback(hObject, eventdata, handles)
handles.eventnum = 8;        guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;end
guidata(hObject, handles);
temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function pushbutton_match_loop_no_Callback(hObject, eventdata, handles)
handles.eventnum = 9;            guidata(hObject, handles);
[state outhandles]= lastloopDataManage(handles);
if(state == 1)        handles = outhandles;      end
guidata(hObject, handles); temp = UpdatelastloopInterface(handles);
% --------------------------------------------------------------------
function Menu_File_Save_Callback(hObject, eventdata, handles)
handles.periodarray(handles.currentperiod.periodnum) = handles.currentperiod;
if(length(handles.periodarray)>0)
        if isequal(handles.FileName,0) | isequal(handles.PathName,0)
                msgbox('There is no data file for inputing!','File Information','warn');
                disp('User selected Cancel');
        else
                tempperiod = handles.periodarray(1);
                flag = writestruct(handles.FileName,handles.PathName,tempperiod,'w');
                for i=2:1:length(handles.periodarray)
                        tempperiod = handles.periodarray(i);
                    flag = writestruct(handles.FileName,handles.PathName,tempperiod,'a');
                end         end
end
% --------------------------------------------------------------------
function Menu_File_Exit_Callback(hObject, eventdata, handles)
guidata(hObject, handles);
uiresume(handles.figure1);
delete(handles.figure1);
```

Data manage function

```
% ------------------------------------------------------------------
%       This function is used to manage the data.
function    [state varargout] = lastloopDataManage(handles)
    state = 1;    updatedhandles = handles;
    eventnum = handles.eventnum;
    switch eventnum
        case -1
        handles.periodarray = [];              handles.currentperiod = PeriodStruct;
        case 0
        [FileName,PathName,FilterIndex] = uiputfile({ ...
                                    '*.fd','new file(*.fd)';'*.*','All file(*.*)'});
        if isequal(FileName,0) | isequal(PathName,0)
                msgbox('There is no data file for inputing!','File
Information','warn');
                disp('User selected Cancel');    handles.FileName = '';
                handles.PathName = '';    handles.FilterIndex = 0;
        else
                handles.FileName = FileName;    handles.PathName = PathName;
                handles.FilterIndex = FilterIndex;
                fid = fopen([handles.PathName handles.FileName],'w');
                fclose(fid);
        end
        case 1
            temp = loaddata;
            if isstruct(temp)
                 if(length(temp) > 0)
                 handles.periodarray = temp; handles.currentperiod = temp(1);
                 end
            else
           msgbox('There is no data file for inputing!','File Information','warn');
            end
        case 2
                handles.currentperiod.periodstate = 1;
```

```
                handles.periodarray(handles.currentperiod.periodnum) = …
            handles.currentperiod;
    case 3
        if(length(handles.currentperiod.eclipsefilename)>0)
            unitdata = [];unitdata(:,1:2) = handles.currentperiod.ratehistory;
            m = handles.currentperiod.propertieshistory;
            unitdata(:,3:4) = m(:,2:3);    unitdata(:,1) = unitdata(:,1)/24;
            outchangedvalue = {};          tempoutchanged = {};
            if(handles.currentperiod.periodnum == 1) initialtime = 0;
            else
                m = handles.periodarray(handles.currentperiod.periodnum-1);
                m = m.ratehistory;    initialtime = m(length(m),1)/24;
            end
            acount = 1;
            if(length(unitdata) > 0)
                for(i=1:1:length(unitdata))
                    tempoutchanged{1} = num2str(unitdata(i,4));
                    tempoutchanged{2} = num2str(unitdata(i,3));
                    if(unitdata(i,2)>0)
                        tempoutchanged{3}    = 'OPEN';
                    else
                        tempoutchanged{3} = 'STOP';
                    end
                    tempoutchanged{4} = num2str(unitdata(i,2));
                    tempoutchanged{5} = num2str(1);
                    if(i==1)
                  tempoutchanged{6} = num2str(unitdata(i,1)-initialtime);
                    else
                  tempoutchanged{6} = num2str(unitdata(i,1)-unitdata(i-1,1));
                    end
                    if(handles.currentperiod.periodnum == 1 && i==1)
                        tempoutchanged = {};
                    else
                        outchangedvalue{acount} = tempoutchanged;
```

```
                    acount=acount+1;
                end
                tempoutchanged = {};
            end
        end
        pathandname = handles.currentperiod.eclipsefilename;
        [PathName,FileName, ext, versn] = fileparts(pathandname);
        keywordsinformation = struct('Keywords','','Unit', '',…
    …'SubKeywords', '','blockKeywords', '','Data',[]);
        Time = keywordsinformation;    Time.Keywords = 'TIME';
        Time.Unit = 'DAYS';    WBHP = keywordsinformation;
        WBHP.Keywords = 'WBHP';    WBHP.Unit = 'PSIA';
        WBHP.SubKeywords = 'PRODUCER';
        WOPR = keywordsinformation;
        WOPR.Keywords = 'WOPR';      WOPR.Unit = 'STB/DAY';
        WOPR.SubKeywords = 'PRODUCER';
        if(length(outchangedvalue) > 0 &    ~isequal(FileName,0))
            initialstep = handles.currentperiod.restartstep;
            PDGData = [outkeywords(1).Data*24
              …outkeywords(2).Data    outkeywords(3).Data];
        end
        if(length(PDGData)>0)
            handles.currentperiod.periodstate = 2;
            handles.currentperiod.simulatedpressure = PDGData(:,1:2);
      handles.eclipsefilename = handles.currentperiod.eclipsefilename ;
            handles.restartstep = restartstepfornext;
            handles.periodarray(handles.currentperiod.periodnum)
          … = handles.currentperiod;
        else
            warndlg('Wrong simulation!!','!! Warning !!')
        end
    end
case 4
    if(length(handles.currentperiod.simulatedpressure)>0)
```

```
                handles.currentperiod.periodstate = 3;
                handles.periodarray(handles.currentperiod.periodnum) =
            …handles.currentperiod;
        end
    case 5
                handles.currentperiod.periodstate = 4;
                handles.periodarray(handles.currentperiod.periodnum) =
            …handles.currentperiod;
                [pathstr, name, ext, versn] = fileparts(handles.eclipsefilename);
                name = '\temp.txt';                pathname = [pathstr name];
                if(handles.currentperiod.periodnum > 1)
        tempperiod = handles.periodarray(handles.currentperiod.periodnum-1);
                    tempperioddata = tempperiod.realpressure;
                    lasttemp = tempperioddata(length(tempperioddata),:);
                    tempcurrent = handles.currentperiod.realpressure;
                    m = [lasttemp ;tempcurrent];              m = m(:,1);
                    delttime = [];
                    for(i=2:1:length(m))    delttime = [delttime m(i)-m(i-1)]; end
                    delttime = delttime/24; fid = fopen(pathname,'wt');
                        for(i=1:1:length(delttime))
                            fprintf(fid,'TSTEP\n');
                            fprintf(fid, '1*%-12.8f\n',delttime(i));
                            fprintf(fid,'/\n');
                        end
                    fclose(fid);
                else
                    m = handles.currentperiod.realpressure;
                    m = m(:,1); delttime = [];
                    for(i=2:1:length(m)) delttime = [delttime m(i)-m(i-1)];    end
                    delttime = delttime/24;
                    fid = fopen(pathname,'wt');
                        for(i=1:1:length(delttime))
                            fprintf(fid,'TSTEP\n');
                            fprintf(fid, '1*%-12.8f\n',delttime(i));
```

```
                    fprintf(fid,'/\n');
                end
            fclose(fid);
        end
        temp = forwardmodelling(handles.currentperiod);
        tempnewparameter = handles.currentperiod.propertieshistory;
        kh = temp(1).Range;                 kk = kh{1};
        kkk = kk{1};                        kh = str2num(kkk);
        tempnewparameter(:,3) = kh;     skin = temp(2).Range;
        sk = skin{1};                       skk = sk{1}
        skin = str2num(skk);                tempnewparameter(:,2) = skin;
        handles.currentperiod.propertieshistory = tempnewparameter;
    case 6
        if(handles.currentperiod.periodnum ~= length(handles.periodarray))
            handles.periodarray(handles.currentperiod.periodnum) =
          …handles.currentperiod;
handles.currentperiod=handles.periodarray(handles.currentperiod.periodnum+1);
            handles.currentperiod.eclipsefilename = handles.eclipsefilename;
            handles.currentperiod.restartstep = handles.restartstep;
            handles.periodarray(handles.currentperiod.periodnum) =
          …handles.currentperiod;
        End
    case 8
        if(length(handles.currentperiod.simulatedpressure)>0)
            handles.currentperiod.periodstate = 5;
            handles.periodarray(handles.currentperiod.periodnum) =
          …handles.currentperiod;
        end
    case 9
        if(length(handles.currentperiod.simulatedpressure)>0)
            handles.currentperiod.periodstate = 4;
            handles.currentperiod.simulatedpressure = [];
            handles.periodarray(handles.currentperiod.periodnum) =
          …handles.currentperiod;
```

```
                end
          otherwise
       end
        if(state == 1)
              updatedhandles = handles;          varargout(1) =    {updatedhandles};
        else
              varargout(1) =    {updatedhandles};
        end
return;
```

Update interface function

```
% --------------------------------------------------------------------
%          This function is used to update the interface.
function    isclass = UpdatelastloopInterface(handles)
periodarray = handles.periodarray;      currentperiod = handles.currentperiod;
isclass = 1;
set(handles.uipanel_control_all,'Visible','off');
set(get(handles.uipanel_control_all,'Children'),'Visible','off');
set(handles.uipanel_for_allfigure,'Visible','off');
set(get(handles.uipanel_for_allfigure,'Children'),'Visible','off');
set(handles.uipanel_match_loop,'Visible','off');
set(get(handles.uipanel_match_loop,'Children'),'Visible','off');
set(handles.uipanel_for_matchingfigure,'Visible','off');
set(get(handles.uipanel_for_matchingfigure,'Children'),'Visible','off');
set(handles.axes_arrow1,'Visible','off'); set(handles.axes_arrow2,'Visible','off');
set(handles.axes_arrow3,'Visible','off'); set(handles.axes_arrow4,'Visible','off');
set(handles.axes_arrow5,'Visible','off');
set(handles.tag_workflow,'Enable','off'); set(handles.tag_help,'Enable','off');
if( length(handles.periodarray)>0 )
      set(handles.tag_workflow,'Enable','on');          set(handles.tag_help,'Enable','on');
end
if(handles.eventnum>=0)
      set(handles.uipanel_control_all,'Visible','on');
      set(get(handles.uipanel_control_all,'Children'),'Visible','on');
      set(get(handles.uipanel_control_all,'Children'),'Enable','off');
```

```
        set(handles.pushbutton_all_loaddata,'Enable','on');
        if(length(periodarray)>0 && length(currentperiod)>0)
            set(handles.text_all_period,'Enable','on');
            set(handles.popupmenu_all_periodname,'Enable','on');
            set(handles.text_all_state,'Enable','on');
            set(handles.popupmenu_all_state,'Enable','on');
            string = '';            sep = '|';            m = periodarray(1);
            string = [string, num2str(m.periodnum), '---',m.periodname];
            for(i=2:1:length(periodarray))
                m = periodarray(i);
               string = [string,sep,num2str(m.periodnum), '---',m.periodname];
            end
            set(handles.popupmenu_all_periodname,'String',string);
            set(handles.popupmenu_all_periodname,'Value',currentperiod.periodnum);
            switch currentperiod.periodstate
                case 1
                    string = 'Normal'; set(handles.popupmenu_all_state,'String',string);
                case 2
              string = 'Run Simulation'; set(handles.popupmenu_all_state,'String',string);
                case 3
                    string = 'Comparision';
                    set(handles.popupmenu_all_state,'String',string);
                case 4
                    string = 'Senstivity Study';
                    set(handles.popupmenu_all_state,'String',string);
                case 5
                    string = 'Next';
                    set(handles.popupmenu_all_state,'String',string);
                otherwise
            end
        end
    end
    if(handles.eventnum>=0)
        set(handles.uipanel_for_allfigure,'Visible','on');
```

```
    set(get(handles.uipanel_for_allfigure,'Children'),'Visible','off');
    if(length(periodarray)>0 && length(currentperiod)>0 )
        set(get(handles.uipanel_for_allfigure,'Children'),'Visible','on');
        axes(handles.axes_all);              temp = [];
        for(i=1:1:length(periodarray))
            temp =[temp; periodarray(i).realpressure];
        end
        plot(handles.axes_all,temp(:,1),temp(:,2),'r-');        hold on;
        xlabel(handles.axes_all,'Time(hours)','Fontsize',8,'FontWeight','bold');
        ylabel(handles.axes_all,'pressure(psi)','Fontsize',8,'FontWeight','bold');
        Title(handles.axes_all,'Real Pressure','Fontsize',10,'FontWeight','bold');
        hold on;              temp = handles.currentperiod.realpressure;
        x1 = temp(1,1);              x2 = temp(length(temp),1);
        temp = get(handles.axes_all,'YLim');
        y1 = temp (1);              y2 = temp (2);
        X = [x1 x1 x2 x2];              Y = [y2 y1 y1 y2];              tcolor = [.7 .7 .7];
        h = patch(X,Y,tcolor,'FaceColor',[0.8,0.8,0.8],...
            'LineStyle','-','EdgeColor',[0.8,0.8,0.8],'LineWidth',1.5);
        hold on;
        set(get(handles.uipanel_for_allfigure,'Children'),'Visible','on');
        axes(handles.axes_all);              temp = [];
        for(i=1:1:length(periodarray)) temp =[temp; periodarray(i).realpressure]; end
        plot(temp(:,1),temp(:,2),'r-');              hold on;
        temp = [];        temparray = handles.periodarray;
        for(i=1:1:length(handles.periodarray))
            temp = [temp ;temparray(i).simulatedpressure];
        end
        if length(temp)>0
            plot(handles.axes_all,temp(:,1),temp(:,2),'g-');
        end
        hold off;
    end
end
if(handles.eventnum>=0)
```

```
        set(handles.uipanel_match_loop,'Visible','on');
        set(handles.pushbutton_match_loop_show_period,'Visible','on');
        set(handles.pushbutton_match_loop_show_period,'string','None');
        set(handles.pushbutton_match_loop_run,'Visible','on');
        set(handles.pushbutton_match_loop_compare,'Visible','on');
        set(handles.pushbutton_match_loop_study,'Visible','on');
        set(handles.pushbutton_match_loop_next,'Visible','on');
        set(handles.text_match_loop_seperate,'Visible','on');
        set(handles.listbox_match_loop_list,'Visible','on');
        set(handles.pushbutton_match_loop_addplot,'Visible','on');
        set(handles.pushbutton1_match_loop_yes,'Visible','on');
        set(handles.pushbutton_match_loop_no,'Visible','on');
        set(handles.pushbutton_match_loop_show_period,'Enable','off');
        set(handles.pushbutton_match_loop_run,'Enable','off');
        set(handles.pushbutton_match_loop_compare,'Enable','off');
        set(handles.pushbutton_match_loop_study,'Enable','off');
        set(handles.pushbutton_match_loop_next,'Enable','off');
        set(handles.text_match_loop_seperate,'Enable','off');
        set(handles.listbox_match_loop_list,'Enable','off');
        set(handles.pushbutton_match_loop_addplot,'Enable','off');
        set(handles.pushbutton1_match_loop_yes,'Enable','off');
        set(handles.pushbutton_match_loop_no,'Enable','off');
        logopicture = imread('workflowarrow1','bmp');
        starearrowup = imread('workflowarrow2','bmp');
        starearrowdown = imread('workflowarrow3','bmp');
        axes(handles.axes_arrow1);        image(logopicture);
        set(handles.axes_arrow1,'Visible','off');        axes(handles.axes_arrow2);
        image(logopicture);      set(handles.axes_arrow2,'Visible','off');
        axes(handles.axes_arrow3);        image(logopicture);
        set(handles.axes_arrow3,'Visible','off');        axes(handles.axes_arrow4);
        image(starearrowup);        set(handles.axes_arrow4,'Visible','off');
        axes(handles.axes_arrow5);        image(starearrowdown);
        set(handles.axes_arrow5,'Visible','off');
if(length(handles.currentperiod.ratehistory) <= 0 || length(handles.periodarray)<=0 )
```

```
            set(handles.pushbutton_match_loop_show_period,'Enable','off');
        else
            set(handles.pushbutton_match_loop_show_period,'ForegroundColor',[0 0 0]);
            set(handles.pushbutton_match_loop_show_period,'Enable','on');
            stringwork = ['period',num2str(handles.currentperiod.periodnum)];
            set(handles.pushbutton_match_loop_show_period,'string',stringwork);
            if(handles.currentperiod.periodstate == 1)
            set(handles.pushbutton_match_loop_show_period,'ForegroundColor',[1 0 0]);
            end
        end
       if(handles.currentperiod.periodstate >= 2    && length(handles.currentperiod…
.eclipsefilename)>0) ||handles.currentperiod.periodstate == 1)
            set(handles.pushbutton_match_loop_run,'ForegroundColor',[0 0 0]);
            set(handles.pushbutton_match_loop_run,'Enable','on');
            if(handles.currentperiod.periodstate == 2)
                set(handles.pushbutton_match_loop_run,'ForegroundColor',[1 0 0]);
            end
        else
            set(handles.pushbutton_match_loop_run,'Enable','off');
        end
       if(handles.currentperiod.periodstate >= 3    && length(handles.currentperiod…
simulatedpressure)>0)|| (handles.currentperiod.periodstate == 2)
            set(handles.pushbutton_match_loop_compare,'ForegroundColor',[0 0 0]);
            set(handles.pushbutton_match_loop_compare,'Enable','on');
            if(handles.currentperiod.periodstate == 3)
              set(handles.pushbutton_match_loop_compare,'ForegroundColor',[1 0 0]);
            end
        else
            set(handles.pushbutton_match_loop_compare,'Enable','off');
        end
        if(handles.currentperiod.periodstate == 4 )
            set(handles.pushbutton_match_loop_study,'ForegroundColor',[0 0 0]);
            set(handles.pushbutton_match_loop_study,'Enable','on');
            if(handles.currentperiod.periodstate == 4)
```

```
            set(handles.pushbutton_match_loop_study,'ForegroundColor',[1 0 0]);
        end
    else
        set(handles.pushbutton_match_loop_study,'Enable','off');
    end
    if(handles.currentperiod.periodstate == 5 && (handles.eventnum == 8 ||
handles.eventnum == 7))
        set(handles.pushbutton_match_loop_next,'ForegroundColor',[0 0 0]);
        set(handles.pushbutton_match_loop_next,'Enable','on');
        if(handles.currentperiod.periodstate == 5)
            set(handles.pushbutton_match_loop_next,'ForegroundColor',[1 0 0]);
        end
    else
        set(handles.pushbutton_match_loop_next,'Enable','off');
    end
end
if(handles.currentperiod.periodstate == 3 || handles.currentperiod.periodstate == 5 )
    set(handles.text_match_loop_seperate,'Enable','on');
    set(handles.listbox_match_loop_list,'Enable','on');
    set(handles.pushbutton_match_loop_addplot,'Enable','on');
    set(handles.pushbutton1_match_loop_yes,'Enable','on');
    set(handles.pushbutton_match_loop_no,'Enable','on');
    stringlist = '';        sep = '|';
    if strcmp(handles.currentperiod.periodname,'Beginshutin')
    stringlist = 'Cartesian Plot'; set(handles.listbox_match_loop_list,'String',stringlist);
    else
        stringlist = ['Original(Cartesian Plot)',sep,'DeltP(Cartesian Plot)',sep,...
        'Semi-log Plot',sep,'log-log plot',sep,'ALL'];
        set(handles.listbox_match_loop_list,'String',stringlist);
    end
end
if( handles.eventnum>=0 )
    switch handles.currentperiod.periodstate
        case 1
```

```
set(get(handles.axes7,'Children'),'Visible','off');
set(handles.uipanel_for_matchingfigure,'Visible','on');
axis(handles.axes8); set(handles.axes8,'Visible','On');
cla(handles.axes8); box on;
m = handles.currentperiod.ratehistory;
plot(m(:,1),m(:,2),'Parent',handles.axes8);
xlabel('Time(hours)','Fontsize',10,'Parent',handles.axes8);
ylabel('Rate(bbl/day)','Fontsize',10,'Parent',handles.axes8);
Title('Production History','Fontsize',12,'Parent',handles.axes8);
axis(handles.axes9)    set(handles.axes9,'Visible','On');
cla(handles.axes9);
if(handles.currentperiod.periodnum>1)
tempperiod = handles.periodarray(handles.currentperiod.periodnum-1);
      tempperioddata = tempperiod.realpressure;
      lasttemp = tempperioddata(length(tempperioddata),:);
      tempcurrent = handles.currentperiod.realpressure;
      m = [lasttemp ;tempcurrent];
else
      m = handles.currentperiod.realpressure;
end
plot(handles.axes9,m(:,1),m(:,2),'r-','LineWidth',2);
xlabel('Time(hours)','Fontsize',10,'Parent',handles.axes9);
ylabel('pressure(psi)','Fontsize',10,'Parent',handles.axes9);
Title('Real Pressure','Fontsize',12,'Parent',handles.axes9);
set(handles.axes10,'Visible','On');              cla(handles.axes10);
m = handles.currentperiod.propertieshistory;
stairs(m(:,1),m(:,2),'-','LineWidth',2,'Parent',handles.axes10);
xlabel('Time(hours)','Fontsize',10,'Parent',handles.axes10);
ylabel('Skin Factor','Fontsize',10,'Parent',handles.axes10);
Title('Skin Factor History','Fontsize',10,'Parent',handles.axes10);
axis(handles.axes11);              set(handles.axes11,'Visible','On');
cla(handles.axes11); m = handles.currentperiod.propertieshistory;
stairs(m(:,1),m(:,3),'-','LineWidth',2,'Parent',handles.axes11);
xlabel('Time(hours)','Fontsize',10,'Parent',handles.axes11);
```

```
        ylabel('Permeability(md)','Fontsize',10,'Parent',handles.axes11);
        Title('Permeability History','Fontsize',10,'Parent',handles.axes11);
    case 2
        if length(handles.currentperiod.simulatedpressure)>0
            set(handles.uipanel_for_matchingfigure,'Visible','on');
          set(get(handles.uipanel_for_matchingfigure,'Children'),'Visible','off');
            set(get(handles.axes8,'Children'),'Visible','off');
            set(get(handles.axes9,'Children'),'Visible','off');
            set(get(handles.axes10,'Children'),'Visible','off');
            set(get(handles.axes11,'Children'),'Visible','off');
            axis(handles.axes7);      set(handles.axes7,'Visible','On');
            cla(handles.axes7);      set(handles.axes7,'XScale','linear');
            set(handles.axes7,'YScale','linear');
            if(handles.currentperiod.periodnum > 1)
        tempperiod = handles.periodarray(handles.currentperiod.periodnum-1);
                tempperioddata = tempperiod.realpressure;
                lasttemp = tempperioddata(length(tempperioddata),:);
                tempcurrent = handles.currentperiod.realpressure;
                temp = handles.currentperiod.realpressure;
                temp = [lasttemp;temp];
          line(temp(:,1),temp(:,2),'Color','r','Marker','o','Parent',handles.axes7);
                temp = handles.currentperiod.simulatedpressure;
                temp = [lasttemp;temp];
          line(temp(:,1),temp(:,2),'Color','g','Marker','*','Parent',handles.axes7);
                xlabel('Time(hours)','Fontsize',12,'Parent',handles.axes7);
                ylabel('pressure(psi)','Fontsize',12,'Parent',handles.axes7);
                Title('Real Pressure and Simulated
             …Pressure','Fontsize',14,'Parent',handles.axes7);
            else
                temp = handles.currentperiod.realpressure;
          line(temp(:,1),temp(:,2),'Color','r','Marker','o','Parent',handles.axes7);
                temp = handles.currentperiod.simulatedpressure;
          line(temp(:,1),temp(:,2),'Color','g','Marker','*','Parent',handles.axes7);
                xlabel('Time(hours)','Fontsize',12,'Parent',handles.axes7);
```

```
                    ylabel('pressure(psi)','Fontsize',12,'Parent',handles.axes7);
                    Title('Real Pressure and Simulated Pressure','Fontsize',
                …14,'Parent',handles.axes7);
                end
            end
        otherwise
            if(handles.eventnum == 7)
                switch get(handles.listbox_match_loop_list,'Value')
                    case 1
                    set(handles.uipanel_for_matchingfigure,'Visible','on');
                set(get(handles.uipanel_for_matchingfigure,'Children'),'Visible','off');
                        set(get(handles.axes8,'Children'),'Visible','off');
                        set(get(handles.axes9,'Children'),'Visible','off');
                        set(get(handles.axes10,'Children'),'Visible','off');
                        set(get(handles.axes11,'Children'),'Visible','off');
                        axis(handles.axes7); set(handles.axes7,'Visible','On');
                        cla(handles.axes7); set(handles.axes7,'XScale','linear');
                        set(handles.axes7,'YScale','linear');
                        if(handles.currentperiod.periodnum <= 1)
                            temp = handles.currentperiod.realpressure;
                            line(temp(:,1),temp(:,2),'Color','r','Marker','o',
                            …'Parent',handles.axes7);
                            temp = handles.currentperiod.simulatedpressure;
                            line(temp(:,1),temp(:,2),'Color','g','Marker','*'
                            …,'Parent',handles.axes7);
                        xlabel('Time(hours)','Fontsize',12,'Parent',handles.axes7);
                        ylabel('pressure(psi)','Fontsize',12,'Parent',handles.axes7);
                            Title('Real Pressure and Simulated
                        …Pressure','Fontsize',14,'Parent',handles.axes7);
                        else
                            tempperioddata = tempperiod.realpressure;
                        lasttemp = tempperioddata(length(tempperioddata),:);
                            tempcurrent = handles.currentperiod.realpressure;
                            temp = [lasttemp ;tempcurrent];
```

```
                                    line(temp(:,1),temp(:,2),'Color','r','Marker',
                                    …'o','Parent',handles.axes7);
                                    temp = handles.currentperiod.simulatedpressure;
                                    temp = [lasttemp; temp];
                line(temp(:,1),temp(:,2),'Color','g','Marker',’*','Parent',handles.axes7);
                    xlabel('Time(hours)','Fontsize',12,'Parent',handles.axes7);
                    ylabel('pressure(psi)','Fontsize',12,'Parent',handles.axes7);
                                    Title('Real Pressure and Simulated Pressure',
                                …'Fontsize',14,'Parent',handles.axes7);
                                end
                            case 2
                                welltestinglastloop(handles);
                            case 3
                                welltestinglastloop(handles);
                            case 4
                                welltestinglastloop(handles);
                            case 5
                                welltestinglastloop(handles);
                            otherwise
                        end                 end       end
end
return;
```

Deconvolution application

```
% ------------------------------------------------------------------
multirate = load('D:\Jun\Deconvolution\programme\change skin factor\table14.txt');
% multi-rate
multirate(:,1) = multirate(:,1)*24;
% ----- fig1 plot original data
figure;
plotyy(multirate(:,1),multirate(:,2),multirate(:,1),multirate(:,3),'plot');
title('multi-Rate and Pressure Data','FontSize',14,'FontWeight','bold');
xlabel('Time(hr)');
beginpoint = 101;
```

```
timebegin = 0.0057899999;
pi = 4014.4138;
%%    step1: preprocessing data
time = multirate(beginpoint:length(multirate),1) - timebegin*24;
multirate = multirate(beginpoint+1:length(multirate),:);
%%    step2: deconvolve
%%%% get the system function
x = multirate(:,3);
y = pi - multirate(:,2);
hcontain = systemresponsef(y,x);
hcontain = [0 hcontain];
ysecond =y(10640) - y(10640:length(y));
xsecond = x(10640:length(x));
hcontainsecond = systemresponsef(ysecond,xsecond);
%hcontainsecond = [0 hcontainsecond];
%%%% get the step signal response
unitcontain = unitresponsef(y,x);
%%    step3: plot
%fig2 plot system response
figure;
% plotyy(time,hcontain,time(1:length(multirate)),multirate(:,2));
plot(time,hcontain,'r.');
hold on;
plot(time(10640:length(x)),hcontainsecond,'*');
title('hcontain','FontSize',14,'FontWeight','bold');
% fig3 plot unit rate response
% pressureunitrate = cumtrapz(hcontain);
pressureunitrate(1) = 0;
for i=2:1:length(hcontain)
      pressureunitrate(i) = pressureunitrate(i-1) + hcontain(i);
end
figure;
plot(time,unitcontain ,'.-');
hold on;
```

```
plot(time,pressureunitrate,'r*-');
title('pressure unit rate response','FontSize',14,'FontWeight','bold');
%%% plot loglog
logvalue = [];
xvalue = [];
time = time';
for(i=1:20:length(hcontain))
    xvalue(i) = time(i);
    logvalue(i) = hcontain(i)*time(i);
end
figure;
loglog(xvalue,logvalue,'.-');
%-----------------%    Deconvolve data with noise %------------------%
% prepare the data
time    = multirate(beginpoint:50:length(multirate),1);
multip = multirate(beginpoint:50:length(multirate),2);
multir = multirate(beginpoint:50:length(multirate),3);
numberofinput = length(multip);
% add noise
noise = wgn(numberofinput,1,2);
pressurenoise = multip + noise*5;
ratenoise = multir    + noise*10;
% plot oriignal data
figure;
plotyy(time,pressurenoise,time,ratenoise);
% deconvolve the pressure data
y = pi-pressurenoise;
x = ratenoise;
hcontain0(1:length(x)) = 0 ;
options=optimset('LargeScale','on','Display','iter');
%% get the impulse response
LB(1:length(hcontain0)) = 0;
UB(1:length(hcontain0)) = inf;
[hcontain1,resnorm,residual]=lsqnonlin(@newfunimpulse,hcontain0,[],[],options,x,y);
```

```
%[hcontain1,resnorm,residual]=lsqnonlin(@newfunimpulse,hcontain0,LB,UB,options,x
,y);
% get the unit sponse
pressureunitrate = [];
pressureunitrate(1) = 0;
for i=2:1:length(hcontain1)
    pressureunitrate(i) = pressureunitrate(i-1) + hcontain1(i);
end
figure;
plot(time,pressureunitrate);
figure;
semilogx(time,pressureunitrate);
figure;
ppp = smooth(pressureunitrate);
plot(time,ppp);
% get the loglog
logvalue = [];
xvalue = [];
time = time';
for(i=1:10:length(hcontain1))
    xvalue(i) = time(i);
    logvalue(i) = hcontain1(i)*time(i);
end
figure;
loglog(xvalue',logvalue,'.-');
figure;
plot(time,hcontain1,'.-');
xlabel('Time(hours)','FontWeight','bold','FontSize',12);
% Create ylabel
ylabel('Pressure(psi)','FontWeight','bold','FontSize',12);
% Create title
title('ImpuLse response','FontSize',14);
%       deconvolution method
function hcontain = systemresponsef(ycontain,xcontain)
```

```
% concolution part h = y/x
%get the first one
hcontain(1) = ycontain(1)/xcontain(1);
% get others
for(i=2:1:length(ycontain))
      for(j =1:1:i-1)
            ycontain(i) = ycontain(i) - xcontain(i-j+1)*hcontain(j);
      end
      hcontain(i) =    ycontain(i)/xcontain(1);
end
return;
% ------------------------------------------------------------------
%% get the step function
function E = newfununittwo(hcontain,x,y)
x=x(:);
y=y(:);
% x = [0;x];
% x = diff(x);
Y(1:length(y)) = 0;
Y = Y';
Y(1)=hcontain(1)*x(1);
temp = [];
for(i=2:1:length(y))
      for(j =1:1:i-1)
            Y(i) = Y(i) + (x(i-j+1) - x(i-j)) * hcontain(j);
      end
      %Y(i) =x(1)*hcontain(i);
end
e(1) = (y(1)-Y(1))^2;
for(i=2:1:length(Y))
      e(i) = e(i-1)+(y(i)-Y(i))^2;
end
E = e(length(e));
return;
```

Well class function

```
%%%
```

```
%                       double Total_C;         //total compressiblity
%                       double Z_Factor;        //Z_factor for gas
%                       struct Gas_compostion; // the composition of gas
%                       sttuct pseudopressure_table;// the result of
%                       presudopressure table
%     methods:
%                   1. the constructor function
%                   function p = WellClass(name,varargin);
function p = wellclass(name,varargin)
% the constractor function of wellclass
switch nargin
 case 0
     % if no input arguments, create a default object
     p.WellName = 'None';
     p.WellType = 'None';       %default is oil well
     p.WellRadius = 0.0;        %Unit feet
     p.WellPDGLocation = 0.0;%unit feet
     p.WellLocation = [];       %[x,y,z]
     p.WellPerforated = 0.0;    %{layer top bottom on/off}
     p.FormationPorosity = 0.0;    % layer porosity
     % dynamic data for well
     p.WellProduction = {[],[],[],[]};
         %       case 1      Realtime_Rate
         %       case 2      Tubeline_Rate
         %       case 3      Cumulated_Rate
         %       case 4      Recoverd_Rate
     p.WellPDGPressure = []; %[time pressure]
     p.WellLog = {};              %array of log
     p.WellWorkOver = {};       %array of WorkOver
     %data for black box model
     p.bbmodel = [];              %black box model [time pressure rate]
     p.matchingpara = [];       %matching parameter
     p.matchingdata = [];       %hisroty matching data [time matching_pressure]
     p.forcastingpara = [];      %forcasting parameter          p.forcastingdata = [];
```

```
%forcasting data [time forcasting_rate forcasting_pressure]
    %% the follow data is the fluid and rock properties
    p.FormationPressure = 0.0;          % the pressure of formation
    p.FormationTemperature = 0.0;    % the temperature of formation
    p.Fluidtype = 0;                % the type of fluid
    p.Fluid_B = 0.0;                % volume of formation factor of fluid
    p.Fluid_C = 0.0;                % Compressibility of fluid
    p.Fluid_U = 0.0;                % visicosity of fluid
    p.Formation_C = 0.0;          % compressibility of formation
    p.Total_C = 0.0;                % total compressiblity
    p.Z_Factor = 0.0;               % Z_factor for gas
    p.Gas_composition = [];    % the composition of gas
    p.pseudopressure_table = [];%    the result of presudopressure table
    p.dataprocesspressure = [];    % the result from data process module
    p = class(p,'wellclass'); %create the well object
case 1
    % if single argument of class wellcalss, return it
    if isa(name,'wellclass')
        p = name;
    else
        p.WellName = name;
        p.WellType = 'Oil';          %default is oil well
        p.WellRadius = 0.0;          %Unit feet
        p.WellPDGLocation = 0.0;%unit feet
        p.WellLocation = [];        %[x,y,z]
        p.WellPerforated = 0.0;    %{layer top bottom on/off}
        p.FormationPorosity = 0.0;    % layer porosity

        p.WellProduction = {[],[],[],[]};
        %         case 1      Realtime_Rate
        %         case 2      Tubeline_Rate
        %         case 3      Cumulated_Rate
        %         case 4      Recoverd_Rate
        p.WellPDGPressure = []; %[time pressure]
```

```
            p.WellLog = {};                %array of log
            p.WellWorkOver = {};        %array of WorkOver
            %data for black box model
            p.bbmodel = [];                %black box model [time pressure rate]
            p.matchingpara = [];%matching parameter
            p.matchingdata = [];%hisroty matching data [time matching_pressure]
            p.forcastingpara = [];      %forcasting parameter
            p.forcastingdata = [];      %forcasting data
            %% the follow data is the fluid and rock properties
            p.FormationPressure = 0.0;          % the pressure of formation
            p.FormationTemperature = 0.0;    % the temperature of formation
            p.Fluidtype = 0;                % the type of fluid (none -- 0 oil --1 ,gas --2 and
water --3)
            p.Fluid_B = 0.0;                % volume of formation factor of fluid
            p.Fluid_C = 0.0;                % Compressibility of fluid
            p.Fluid_U = 0.0;                % visicosity of fluid
            p.Formation_C = 0.0;          % compressibility of formation
            p.Total_C = 0.0;                % total compressiblity
            p.Z_Factor = 0.0;               % Z_factor for gas
            p.Gas_composition = [];    % the composition of gas
            p.pseudopressure_table = [];%    the result of presudopressure table
            p.dataprocesspressure = [];    % the result from data process module
            p = class(p,'wellclass');
        disp([inputname(1) ' is not a wellclass object']);
    otherwise
            % if single argument of class wellcalss, return it
            % create object using specified arguments
            p.WellName = name;
            p.WellType = 'oil';          %default is oil well
            p.WellRadius = 0.0;          %Unit feet
            p.WellPDGLocation = 0.0;%unit feet
            p.WellLocation = [];        %[x,y,z]
            p.WellPerforated = 0.0;
            p.FormationPorosity = 0.0;    % layer porosity
```

```
p.WellProduction = {[],[],[],[]};
%       case 1      Realtime_Rate
%       case 2      Tubeline_Rate
%       case 3      Cumulated_Rate
%       case 4      Recoverd_Rate
p.WellPDGPressure = []; %[time pressure]
p.WellLog = {};             %array of log
p.WellWorkOver = {};        %array of WorkOver
%data for black box model
p.bbmodel = [];             %black box model [time pressure rate]
p.matchingpara = [];
p.matchingdata = [];
p.forcastingpara = [];      %forcasting parameter
p.forcastingdata = [];
%% the follow data is the fluid and rock properties
p.FormationPressure = 0.0;          % the pressure of formation
p.FormationTemperature = 0.0;    % the temperature of formation
p.Fluidtype = 0;
p.Fluid_B = 0.0;            % volume of formation factor of fluid
p.Fluid_C = 0.0;            % Compressibility of fluid
p.Fluid_U = 0.0;            % visicosity of fluid
p.Formation_C = 0.0;        % compressibility of formation
p.Total_C = 0.0;            % total compressiblity
p.Z_Factor = 0.0;           % Z_factor for gas
p.Gas_composition = [];    % the composition of gas
p.pseudopressure_table = [];%    the result of presudopressure table
flag = [];
% the follow arguments
for k = 1:2:length(varargin)
    switch varargin{k}
        case 'WellType'
            p.WellType = varargin{k+1};             %default is oil well
            flag = [flag int8((k+1)/2)];
        case 'WellRadius'
```

```
        p.WellRadius = varargin{k+1};              %Unit feet
        flag = [flag int8((k+1)/2)];
    case 'WellPDGLocation'
        p.WellPDGLocation = varargin{k+1};    %unit feet
        flag = [flag int8((k+1)/2)];
    case 'WellLocation'
        p.WellLocation = varargin{k+1};          %[x,y,z]
        flag = [flag int8((k+1)/2)];
    case 'WellPerforated'
        p.WellPerforated = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'FormationPorosity'
        p.FormationPorosity = varargin{k+1};      %{layer porosity }
        flag = [flag int8((k+1)/2)];
    case 'WellProduction'
        p.WellProduction = varargin{k+1};    %
        flag = [flag int8((k+1)/2)];
    case 'WellPDGPressure'
        p.WellPDGPressure = varargin{k+1}; %[time pressure]
        flag = [flag int8((k+1)/2)];
    case 'WellLog'
        p.WellLog = varargin{k+1};                %array of log
        flag = [flag int8((k+1)/2)];
    case 'WellWorkOver'
        p.WellWorkOver = varargin{k+1};        %array of WorkOver
        flag = [flag int8((k+1)/2)];
    %data for black box model
    case 'bbmodel'
         p.bbmodel = varargin{k+1};
         flag = [flag int8((k+1)/2)];
    case 'matchingpara'
        p.matchingpara = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'matchingdata'
```

```
        p.matchingdata = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'forcastingpara'
        p.forcastingpara = varargin{k+1};      %forcasting parameter
        flag = [flag int8((k+1)/2)];
    case 'forcastingdata'
        p.forcastingdata = varargin{k+1};      %forcasting data
        flag = [flag int8((k+1)/2)];
    %% the follow data is the fluid and rock properties
    case 'FormationPressure'
        p.FormationPressure = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'FormationTemperature'
        p.FormationTemperature = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'Fluidtype'
        p.Fluidtype = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'Fluid_B'
        p.Fluid_B = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'Fluid_C'
        p.Fluid_C = varargin{k+1};      % Compressibility of fluid
        flag = [flag int8((k+1)/2)];
    case 'Fluid_U'
        p.Fluid_U    = varargin{k+1};      % visicosity of fluid
        flag = [flag int8((k+1)/2)];
    case 'Formation_C'
        p.Formation_C    = varargin{k+1};
        flag = [flag int8((k+1)/2)];
    case 'Total_C'
        p.Total_C    = varargin{k+1};      % total compressiblity
        flag = [flag int8((k+1)/2)];
    case 'Z_Factor'
```

```
                    p.Z_Factor    = varargin{k+1};      % Z_factor for gas
                    flag = [flag int8((k+1)/2)];
                case 'Gas_composition'
                    p.Gas_composition    = varargin{k+1};
                    flag = [flag int8((k+1)/2)];
                case 'pseudopressure_table'
                    p.pseudopressure_table    = varargin{k+1};
                    flag = [flag int8((k+1)/2)];
                case 'dataprocesspressure'
                    p.dataprocesspressure    = varargin{k+1};
                  %    the result of presudopressure table
                    flag = [flag int8((k+1)/2)];
                otherwise
                    disp([varargin{k} ' is not a wellclass element']);
            end
        end
        %create the well object
        p = class(p,'wellclass'); %create the well object
end
% ---------------------------------------------------------------------
% this function is used to save the wellcalss
%    function filefid = savewell(wellh,filefid)
%    input:
%                wellh      ---- wellclass object
%                filefid ---- file object
%    output:
%                filefid ---- file object
function filefid = savewell(wellh,filefid)
%1. save the wellname
    fprintf(filefid,'WellName:\n ');
    fprintf(filefid,'%s\n',wellh.WellName);
%2. save the welltype
    fprintf(filefid,'WellType:\n ');
    fprintf(filefid,'%s\n',wellh.WellType);
```

```
%3. save the well radius
    fprintf(filefid,'WellRadius(feet):\n ');
    fprintf(filefid,'%f\n',wellh.WellRadius);
%3.0 save the formation thickness
    fprintf(filefid,'WellPerforated(feet):\n ');
    fprintf(filefid,'%f\n',wellh.WellPerforated);
%3.1 save the formation porosity
    fprintf(filefid,'Porosity:\n ');
    fprintf(filefid,'%f\n',wellh.FormationPorosity);
%3.F.R save production rate
    % the type of fluid (none -- 0 oil --1 ,gas --2 and water --3)
    fprintf(filefid,'Fluid Type:\n ');
    fprintf(filefid,'%d\n',wellh.Fluidtype);
    %volume of formation factor of fluid
    fprintf(filefid,'volume of formation factor of fluid:\n ');
    fprintf(filefid,'%f\n',wellh.Fluid_B);
    %Compressibility of fluid
    fprintf(filefid,'Compressibility of fluid:\n ');
    fprintf(filefid,'%f\n',wellh.Fluid_C);
    %visicosity of fluid
    fprintf(filefid,'visicosity of fluid:\n ');
    fprintf(filefid,'%f\n',wellh.Fluid_U);
    %compressibility of formation
    fprintf(filefid,'compressibility of formation:\n ');
    fprintf(filefid,'%f\n',wellh.Formation_C);
    %total compressiblity
    fprintf(filefid,'total compressiblity:\n ');
    fprintf(filefid,'%f\n',wellh.Total_C);
    %Z_factor for gas
    fprintf(filefid,'Z_factor for gas:\n ');
    fprintf(filefid,'%f\n',wellh.Z_Factor);
    %the pressure of formation
    fprintf(filefid,'the pressure of formation:\n ');
    fprintf(filefid,'%f\n',wellh.FormationPressure);
```

```
        % the temperature of formation
        fprintf(filefid,'the temperature of formation:\n ');
        fprintf(filefid,'%f\n',wellh.FormationTemperature);
%4. save PDG pressure
        [m,n] = size(wellh.WellPDGPressure);
        fprintf(filefid,'PDG--pressure:\n    %d \n',m);
        fprintf(filefid,'Number      Time(hours)          Pressure(psi)\n');
        if(m ~= 0)
            for(i = 1:1:m)
               fprintf(filefid,'%12d        ',wellh.WellPDGPressure(i,1));
               for(j = 2:1:n)
                  fprintf(filefid,'%12.6f        ',wellh.WellPDGPressure(i,j));
               end
               fprintf(filefid,'\n');
            end
        end
%5. save production rate
  tempdata = wellh.WellProduction;
  for i=1:1:4
         temp = tempdata{i};
         [m,n] = size(temp);
         switch i
              %         case 1      Realtime_Rate
              case 1
                  fprintf(filefid,'Rate%d:\n    %d \n',i,m);
                  fprintf(filefid,'Number      Time(hours)          Rate(mmscf/day)\n');
              %         case 2      Tubeline_Rate
              case 2
                  fprintf(filefid,'Tubline Rate%d:\n    %d \n',i,m);
                  fprintf(filefid,'Number      Time(hours)          Rate(mmscf/day)\n');

              %         case 3      Cumulated_Rate
              case 3
                  fprintf(filefid,'Daily Rate%d:\n    %d \n',i,m);
```

```
                fprintf(filefid,'Time(hours)                    GasRate(mmscf/day)
oilRate(bbl/day)\n');
            %       case 4      Recoverd_Rate
            case 4
                fprintf(filefid,'Rate%d:\n    %d \n',i,m);
                fprintf(filefid,'Time(hours)          gas    Rate(mmscf/day)        oil
Rate(bbl/day)\n');
            otherwise
        end

        if(m ~= 0)
            for(i = 1:1:m)
              fprintf(filefid,'%12d       ',temp(i,1));
              for(j = 2:1:n)
                fprintf(filefid,'%12.6f       ',temp(i,j));
              end
              fprintf(filefid,'\n');
            end
      end
  end
  %6. save processing result
      [m,n] = size(wellh.dataprocesspressure);
      fprintf(filefid,'PDG--the result of data processing:\n    %d \n',m);
      fprintf(filefid,'Number      Time(hours)         Pressure(psi)\n');
      if(m ~= 0)
            for(i = 1:1:m)
              fprintf(filefid,'%12d       ',wellh.dataprocesspressure(i,1));
              for(j = 2:1:n)
                fprintf(filefid,'%12.6f       ',wellh.dataprocesspressure(i,j));
              end
              fprintf(filefid,'\n');
            end
      end
return;
```

```
% ------------------------------------------------------------------
% this function is used to load the wellcalss
%    function filefid = loadwell(wellh,filefid)
%    input:
%                  wellh      ---- wellclass object
%                  filefid ---- file object
%    output:
%                  wellh    ---- wellclass object
function wellh = loadwell(wellh,filefid)
      tline = fgetl(filefid);
%1. load the wellname
     tline =    fgetl(filefid);      tline = fgetl(filefid);
     wellh = set(wellh,'WellName',tline);
%2. load the welltype
      tline =    fgetl(filefid);    tline =    fgetl(filefid);    wellh.WellType = tline;
%3. load the well radius
      tline =    fgetl(filefid);       tline =    fgetl(filefid);
      wellh.WellRadius = str2num(tline);
%3.0 load the WellPerforated
      tline =    fgetl(filefid);       tline =    fgetl(filefid);
      wellh.WellPerforated = str2num(tline);
%3.1 load the formation porosity
      tline =    fgetl(filefid);      tline =    fgetl(filefid);
      wellh.FormationPorosity = str2num(tline);
 %3.2 the type of fluid (none -- 0 oil --1 ,gas --2 and water --3)
      tline   =     fgetl(filefid);          tline   =     fgetl(filefid);          wellh.Fluidtype   =
str2num(tline);
 %3.3 volume of formation factor of fluid
      tline =    fgetl(filefid);       tline =    fgetl(filefid);
      wellh.Fluid_B = str2num(tline);
 %3.4 Compressibility of fluid
      tline =    fgetl(filefid);       tline =    fgetl(filefid);
      wellh.Fluid_C = str2num(tline);
 %3.5 visicosity of fluid
```

```
    tline =    fgetl(filefid);       tline =    fgetl(filefid);
    wellh.Fluid_U = str2num(tline);
%3.6 compressibility of formation
    tline =    fgetl(filefid);       tline =    fgetl(filefid);
    wellh.Formation_C = str2num(tline);
    %3.7 total compressiblity
    tline =    fgetl(filefid);       tline =    fgetl(filefid);
    wellh.Total_C = str2num(tline);
%3.8 Z_factor for gas
    tline =    fgetl(filefid);       tline =    fgetl(filefid);
    wellh.Z_Factor = str2num(tline);
%3.9 the pressure of formation
    tline =    fgetl(filefid);    tline =    fgetl(filefid);
    wellh.FormationPressure = str2num(tline);
%3.10 the temperature of formation
    tline =    fgetl(filefid);         tline =    fgetl(filefid);
    wellh.FormationTemperature = str2num(tline);
%4. load PDG pressure
    tline = fgetl(filefid);         m = fscanf(filefid,'%d');
    tline = fgetl(filefid);
    %read the data
    a = [];
    if(m ~= 0)
        a = fscanf(filefid,'%d %f %f',[3 m]);
        tline = fgetl(filefid);
    end
    wellh.WellPDGPressure = a';
%5. load production rate
mcell =    {[],[],[],[]};
for i=1:1:4
    tline = fgetl(filefid);
    m = fscanf(filefid,'%d');
    tline = fgetl(filefid);
    %read the data
```

```
        a = [];
        if(m ~= 0)
            if(i==4)
                a = fscanf(filefid,'%f %f %f',[3 m]);            tline = fgetl(filefid);
            else
                a = fscanf(filefid,'%d %f %f',[3 m]);            tline = fgetl(filefid);
            end
        end
        mcell{i} = a';
    end
wellh.WellProduction = mcell;
    %4. load PDG pressure
tline = fgetl(filefid); m = fscanf(filefid,'%d');
tline = fgetl(filefid);
%read the data
a = [];
if(m ~= 0)
        a = fscanf(filefid,'%d %f %f',[3 m]);        tline = fgetl(filefid);
end
wellh.dataprocesspressure = a';
return;
% ------------------------------------------------------------------
function a = set(a,varargin)
% SET Set asset properties and return the updated object
propertyArgIn = varargin;
while length(propertyArgIn) >= 2,
        prop = propertyArgIn{1};
        val = propertyArgIn{2};
        propertyArgIn = propertyArgIn(3:end);
      switch prop
          case 'WellName'                a.WellName = val;
          case 'WellType'                a.WellType= val;
          case 'WellRadius'              a.WellRadius= val;
          case 'WellPDGLocation'         a.WellPDGLocation= val;
```

```
            case 'WellLocation'                a.WellLocation= val;
            case 'WellPerforated'              a.WellPerforated= val;
            case 'WellProduction'              a.WellProduction= val;
            case 'WellPDGPressure'             a.WellPDGPressure= val;
            case 'WellLog'                     a.WellLog= val;
            case 'WellWorkOver'                a.WellWorkOver= val;
            case 'bbmodel'                     a.bbmodel    = val;
            case 'matchingpara'                a.matchingpara = val;
            case 'matchingdata'                a.matchingdata = val;
            case 'forcastingpara'              a.forcastingpara = val;
            case 'forcastingdata'              a.forcastingdata = val;
            case 'FormationPorosity'           a.FormationPorosity = val;
            case 'Fluidtype'                a.Fluidtype = val;
            case 'Fluid_B'                  a.Fluid_B = val;
            case 'Fluid_C'                  a.Fluid_C = val;
            case 'Fluid_U'                  a.Fluid_U = val;
            case 'Formation_C'              a.Formation_C = val;
            case Total_C'                   a.Total_C = val;
            case 'Z_Factor'                 a.Z_Factor = val;
            case 'Gas_composition'          a.Gas_composition = val;
            case 'pseudopressure_table'     a.pseudopressure_table = val;
            case 'FormationPressure'        a.FormationPressure = val;
            case 'FormationTemperature'     a.FormationTemperature = val;
          case    'dataprocesspressure'     a.dataprocesspressure = val;
            otherwise
                error('Asset properties');
        end
end
return;
% ------------------------------------------------------------------
function val = get(p, propName)
% GET Get asset properties from the specified object
% and return the value
switch propName
```

```
        case 'WellName'                 val = p.WellName;
        case 'WellType'                 val = p.WellType;
        case 'WellRadius'                val = p.WellRadius;
        case 'WellPDGLocation'          val = p.WellPDGLocation;
        case 'WellLocation'             val = p.WellLocation;
        case 'WellPerforated'           val = p.WellPerforated;
        case 'WellProduction'            val = p.WellProduction;
        case 'WellPDGPressure'           val = p.WellPDGPressure;
        case 'WellLog'                   val = p.WellLog;
        case 'WellWorkOver'              val = p.WellWorkOver;
        case 'bbmodel'                   val = p.bbmodel;
        case 'matchingpara'              val = p.matchingpara;
        case 'matchingdata'              val = p.matchingdata;
        case 'forcastingpara'            val = p.forcastingpara;
        case 'forcastingdata'            val = p.forcastingdata;
        case 'FormationPorosity'         val = p.FormationPorosity;
        case   'Fluidtype'               val = p.Fluidtype;
        case   'Fluid_B'                 val = p.Fluid_B;
        case   'Fluid_C'                 val = p.Fluid_C;
        case   'Fluid_U'                 val = p.Fluid_U;
        case   'Formation_C'             val = p.Formation_C;
        case   'Total_C'                  val = p.Total_C;
        case   'Z_Factor'                 val = p.Z_Factor;
        case   'Gas_composition'          val = p.Gas_composition;
        case   'pseudopressure_table'     val = p.pseudopressure_table;
        case   'FormationPressure'        val = p.FormationPressure;
        case   'FormationTemperature'     val = p.FormationTemperature;
         otherwise
           error([propName,' Is not a valid asset property']);
end
% ------------------------------------------------------------------
function val = subsref(p,index)
%SUBSREF Define field name indexing for well objects
switch index.type
```

```
case '()'
    switch index.subs{:}
      case 1          val = p.WellName;              %well nale
      case 2          val = p.WellType;              %default is oil well
      case 3          val = p.WellRadius;            %Unit feet
      case 4          val = p.WellPDGLocation;   %unit feet
      case 5          val = p.WellLocation;        %[x,y,z]
      case 6          val = p.WellPerforated;    %{layer top bottom on/off}
      case 7          val = p.WellProduction;    %
      case 8          val = p.WellPDGPressure; %[time pressure]
      case 9          val = p.WellLog;               %array of log
      case 10         val = p.WellWorkOver;      %array of WorkOver
      case 11         val = p.bbmodel;     %black box model [time pressure rate]
      case 12         val = p.matchingpara;        %matching parameter
      case 13         val = p.matchingdata;        %hisroty matching data
      case 14         val = p.forcastingpara;    %forcasting parameter
      case 15         val = p.forcastingdata;
      case 16         val = p.FormationPorosity;     %formation porosity
      case 17         val = p.Fluidtype;             % the type of fluid
      case 18         val = p.Fluid_B;           % volume of formation factor of fluid
      case 19         val = p.Fluid_C;               % Compressibility of fluid
      case 20         val = p.Fluid_U;               % visicosity of fluid
      case 21         val = p.Formation_C;       % compressibility of formation
      case 22         val = p.Total_C;               % total compressiblity
      case 23         val = p.Z_Factor;              % Z_factor for gas
      case 24         val = p.Gas_composition;   % the composition of gas
      case 25       val = p.pseudopressure_table;% presudopressure table
      case 26       val = p.FormationPressure;% the pressure of formation
      case 27       val = p.FormationTemperature;% the temperature of formation
    otherwise       error('Index out of range')
    end
case '.'
    switch index.subs
        case 'WellName'          val = p.WellName;              %well nale
```

```
        case 'WellType'                val = p.WellType;                %default is oil well
        case 'WellRadius'              val = p.WellRadius;              %Unit feet
        case 'WellPDGLocation'     val = p.WellPDGLocation;    %unit feet
        case 'WellLocation'            val = p.WellLocation;          %[x,y,z]
        case 'WellPerforated'          val = p.WellPerforated; %{layer top bottom on/off}
        case 'WellProduction'          val = p.WellProduction;    %
        case 'WellPDGPressure'     val = p.WellPDGPressure; %[time pressure]
        case 'WellLog'                 val = p.WellLog;
        case 'WellWorkOver'            val = p.WellWorkOver;
        case 'bbmodel'                 val = p.bbmodel;
        case 'matchingpara'            val = p.matchingpara;
        case 'matchingdata'            val = p.matchingdata;
        case 'forcastingpara'          val = p.forcastingpara;
        case 'forcastingdata'          val = p.forcastingdata;
        case 'FormationPorosity'    val = p.FormationPorosity;
        case 'Fluidtype'               val = p.Fluidtype;
        case 'Fluid_B'                 val = p.Fluid_B;
        case 'Fluid_C'                 val = p.Fluid_C;
        case    'Fluid_U'              val = p.Fluid_U;
        case    'Formation_C'          val = p.Formation_C;
        case    'Total_C'              val = p.Total_C;
        case    'Z_Factor'             val = p.Z_Factor;
        case 'Gas_composition'    val = p.Gas_composition;
        case    'pseudopressure_table'      val = p.pseudopressure_table;
        case    'FormationPressure'         val = p.FormationPressure;
        case    'FormationTemperature'    val = p.FormationTemperature;
        case    'dataprocesspressure'      val = p.dataprocesspressure;
        otherwise              error('Invalid field name')
    end
case '{}'
    error('Cell array indexing not supported by asset objects')
end
% -------------------------------------------------------------------
function display(p)
```

```
% WELLCLASS/DISPLAY Command window display of a WELLCLASS
disp(['1.wellname    ',p.WellName]);
disp(['2.welltype ',p.WellType]);      disp(['3.WellRadius(feet)']);
disp(p.WellRadius);                   disp(['4.WellPDGLocation(feet) ']);
disp(p.WellPDGLocation);              disp(['5.WellLocation[x,y,z] ']);
disp(p.WellLocation);                 disp(['6.WellPerforated ']);
disp(p.WellPerforated);               disp(['7.WellProduction ']);
disp(p.WellProduction);               disp(['8.WellPDGPressure ']);
disp(p.WellPDGPressure);              disp(['9.WellLog ']);
disp(p.WellLog);                      disp(['10.WellWorkOver ']);
disp(p.WellWorkOver);                 disp(['11.Black Box Model']);
disp(p.bbmodel);                      disp(['12.History matching parameter']);
disp(p.matchingpara);                 disp(['13.hisroty matching data']);
disp(p.matchingdata);                 disp(['14.forcasting parameter']);
disp(p.forcastingpara);               disp(['15. forcasting data']);
disp(p.forcastingdata);               disp(['16. formation porosity']);
disp(p.FormationPorosity);            disp(['17. the type of fluid ']);
disp(p.Fluidtype);                    disp(['18. volume of formation factor ']);
disp(p.Fluid_B);                      disp(['19. Compressibility of fluid ']);
disp(p.Fluid_C);                      disp(['20. visicosity of fluid ']);
disp(p.Fluid_U);                      disp(['21.compressibility of formation ']);
disp(p.Formation_C);                  disp(['22. total compressiblity    ']);
disp(p.Total_C);                      disp(['23. Z_factor for gas ']);
disp(p.Z_Factor);                    disp(['24. Gas_composition ']);
disp(p.Gas_composition);              disp(['25. pseudopressure_table']);
disp(p.pseudopressure_table);         disp(['26. the pressure of formation ']);
disp(p.FormationPressure);            disp(['27.the temperature of formation ']);
disp(p.FormationTemperature);     disp(['28.the result from data processing module']);
disp(p.dataprocesspressure);
```

References

[1] A. Bond, and T. Kragas, and S. Mathias, “Real-Time Optical Monitoring of Bottomhole Pressure During Extreme Overbalanced Perforating and Production Testing of a Remote Alaskan Exploration Well”, International Symposium and Exhibition on Formation Damage Control, Lafayette, Louisiana U.S.A., 15-17 February 2006

[2] A.F. van Everdingen, and W. Hurst, "The Application of the Laplace Transformation to Flow Problems in Reservoirs", Petroleum Transactions, AIME 186,305–324 (1949)

[3] Ahn Sanghui, and Roland N. Horne, “Analysis of Permanent Downhole Gauge Data by Cointerpretation of Simultaneous Pressure and Flow Rate Signals”, SPE Annual Technical Conference and Exhibition, Denver, Colorado, USA, 21-24 September 2008

[4] Alain C. Gringarten, P. Bourdet Dominique, A. Landel Pierre, J.Kniazeff Vladimir, “A Comparison Between Different Skin and Wellbore Storage Type-curves for Really-Time Transient Analysis”, SPE Annual Technical Conference and Exhibition, Las Vegas, Nevada, 23-26 September 1979

[4] Alain C. Gringarten, Thomas von Schroeter, Trond Rolfsvaag, John Bruner, “Use of Downhole Permanent Pressure Gauge Data to Diagnose Production Problems in a North Sea Horizontal Well”, SPE Annual Technical Conference and Exhibition, Denver, Colorado, 5-8 October 2003

[6] Alain C. Gringarten, “From Straight Lines to Deconvolution: The Evolution of the State of the Art in Well Test Analysis”, SPE Reservoir Evaluation & Engineering, Volume 11, Number 1, 41-62 (2008)

[7] Andy Bond, Curt Blount,Tor Kragas, Steve Mathias, “Use of a Fiber Optic Pressure/Temperature Gauge in an Exploration Well to Minimize Formation Damage Potential and Reduce Costs During Production Testing”, SPE Annual Technical

Conference and Exhibition, Houston, Texas, 26-29 September 2004

[8] Barys Güyagüler, Roland N. Horne, Eric Tauzin, "Automated Reservoir Model Selection in Well-Test Interpretation", SPE Reservoir Evaluation & Engineering, Volume 6, Number 2, 100-107 (2003)

[9] B. Baygü, F.J. Kuchuk, "Deconvolution Under Normalized Autocorrelation Constraints", SPE Journal, Volume 2, Number 3, 246-253, (1997)

[10] B. Guyanguler, Roland N. Horne, Eric Tauzin, "Automated Reservoir Model Selection in Well Test Interpretation", SPE Reservoir Evaluation & Engineering, 100-107 (2003)

[11] B. Izgec, M.E. Cribbs, S.V. Pace, and C.S. Kabir, "Placement of Permanent Downhole Pressure Sensors in Reservoir Surveillance", EUROPEC/EAGE Conference and Exhibition, London, U.K., 11-14 June 2007

[12] B.K. Drakeley, and Svein Omdal, "Fiber Optics Sensing Systems for Subsea Applications—Sensing Capabilities, Applications, and the Challenges Being Faced in Order to Provide Reliable Transmission of Data for Online Reservoir Management", Intelligent Energy Conference and Exhibition, Amsterdam, The Netherlands, 25-27 February 2008

[13] B.L. Gingerich, and P.G. Brusius, and I.M. Maclean, "Reliable Electronics for High-Temperature Downhole Applications", SPE Annual Technical Conference and Exhibition, Houston, Texas, 3-6 October 1999

[14] B. lzgec; M.E. Cribbs, S.V. Pace, et al., "Placement of Permanent Downhole Pressure Sensors in Reservoir Surveillance", SPE Europec/EAGE Annual Conference and Exhibition, London, UK, 11-14 June 2007

[15] B.P. Champion, and I.R. Searle and R.K. Pollard," Clair Field: Reducing Uncertainty in Reservoir Connectivity During Reservoir Appraisal—A First-Time Application of a New Wireless Pressure-Monitoring Technology in an Abandoned Subsea Appraisal Well" Offshore Europe, Aberdeen, Scotland, U.K., 4-7 September 2007

[16] B. Smith, SPE, and M. Hall, and A. Franklin, SPE, E.S. Johansen, SPE, and Ö.H. Ünalmis," Field-Wide Deployment of In-Well Optical Flowmeters and Pressure/Temperature Gauges at Buzzard Field", Intelligent Energy Conference and Exhibition, Amsterdam, The Netherlands, 25-27 February 2008

[17] B. Widarsono and H. Atmoko, lemigas, "Application of Fuzzy Logic for Determining Production Allocation in Commingle Production Wells", SPE Asian

Pacific Oil & Gas Conference and Exhibition, Jakarta, Indonesia, 5-7April, 2005

[18] C. Amudo, J. Turner, J. Frewin, and T.C. Kgogo, PetroSA, and A.C. Gringarten, "Integration of Well Test Deconvolution Analysis and Detailed Reservoir Modelling in 3D Seismic Data Interpretation: A Case Study", SPE Europec/EAGE Annual Conference and Exhibition, Vienna, Austria, 12-15 June 2006

[19] C.A. Ehlig-Economides and P. Valko, I.S. Dyashev, Sibneft; and M.J. Economides, "Pressure Transient and Production Data Analysis for Hydraulic Fracture Treatment Evaluation", SPE Russian Oil and Gas Technical Conference and Exhibition, Moscow, Russia, 3-6 October 2006

[20] C.E. Shepherd, P. Neve, D.C. Wilson,"Use and Application of Permanent Downhole Pressure Gauges in the Balmoral Field and Satellite Structures". SPE Production Engineering, 6 (3), 271–276 (1991)

[21] C.J. Airlie, and Z.R. Lemanczyk, "Intelligent Asset Management: Successful Integration of Modeling Tools and Workflow Processes", SPE Asia Pacific Conference on integrated modeling for Asset management, Malaysia, Mar 2004

[22] Da Prat, Eliseche, L., Iriarte, "A New Method for Evaluating Layer Productivity Using a Permanent Monitoring System", SPE International Oil and Gas Conference and Exhibition in China, Beijing, China, 2-6 November 1998

[23] Dario Viberti, Francesca Verga, and Paolo Delbosco, et al., "An Improved Treatment of Long-term Pressure Data for Capturing Information", SPE Reservoir Evaluation & Engineering, 359-366 (2007)

[24] De Oliveira Silva, E.T. Kato, "Reservoir Management Optimization Using Permanent Downhole Gauge Data". SPE Annual Technical Conference and Exhibition, Houston, 26–29 September 2004

[25] D. Ilk, P.P. Valko, and T.A. Blasingame," Deconvolution of Variable-Rate Reservoir Performance Data Using B-Splines", SPE Annual Technical Conference and Exhibition, 9-12 October 2005, Dallas, Texas

[26] D. Ilk, P.P. Valko, and T.A. Blasingame, "Deconvolution of Variable-Rate Reservoir Performance Data Using B-Splines", SPE Reservoir Evaluation & Engineering, Volume 9, Number 4, 582-595 (2006)

[27] D. Ilk, P.P. Valkó, and T.A. Blasingame, "A Deconvolution Method Based on Cumulative Production for Continuously Measured Flow rate and Pressure Data", Eastern Regional Meeting, Lexington, Kentucky USA, 17-19 October 2007

[28] D.L. Katz, M.R. Tek, S.C. Jones, "A Generalized model for predicting the

performance of gas reservoir subject to water drive", Paper SPE 428 presented at the SPE Annual Meeting, Los Angeles, 7-10 Oct 1962

[29] D.M. Chorneyko, "Real-Time Reservoir Surveillance Utilizing Permanent Downhole Pressures—An Operator's Experience", SPE Annual Technical Conference and Exhibition, San Antonio, Texas, 24–27 September 2006

[30] Dominique Bourdet, "Well Test Analysis: The Use of Advanced Interpretation Models", ELSEVIER, (2002)

[31] Dominique Bourdet, J.A. Ayoub, Y.M. Pirard, Flopetrol-Johnston, "Use of Pressure Derivative in Well Test Interpretation", SPE Formation Evaluation, 293-302 (1989)

[32] D.R. Horner, "Pressure Build-ups in wells", Third world petroleum congress, The Hague, Section two, 503-523 (1951)

[33] D.S. Bustamante, and G.D. Monson, "Understanding Reservoir Performance and Uncertainty Using a Multiple History Matching Process", SPE Annual Technical Conference and Exhibition, Dallas, Texas, U.S.A., 9-12 October 2005

[34] Du Kuifu, "Use of Advanced Pressure Transient Analysis Techniques To Improve Drainage Area Calculations and Reservoir Characterisation: Field Case Studies", Offshore Europe, Aberdeen, Scotland, U.K., 4-7 September 2007

[35] Du Kuifu, "The Determination of Tested Drainage Area and Reservoir Characterisation from Entire Well-Test History By Deconvolution and Conventional Pressure-Transient Analysis Techniques", SPE Annual Technical Conference and Exhibition, Denver, Colorado, USA, 21-24 September 2008

[36] D. Viberti, F. Verga, and P. Delbosco, Politecnico di Torino, "A New Approach for Capitalizing on Continuous Downhole Pressure Data", SPE Annual Technical Conference and Exhibition, Dallas, Texas, 9-12 October 2005

[37] E.J. Coludrovich, J.D. McFadden, M.R. Palke, W.R. Roberts, and L.J. Robson, "The Boris Field Well Management Philosophy—The Application of Permanent Downhole Flow meters to Pressure Transient Analysis: An Integrated Approach",SPE Annual Technical Conference and Exhibition, Houston,26–29 Sep2004

[38] F. Gonzalez-Tamez, Camacho Velazquez, "Truncation De-Noise in Transient Pressure tests", SPE Annual Technical Conference and Exhibition, Houston, Texas, 3-6 October 1999

[39] F.P.T. Silva, J.R.P. Rodrigues, P.L.B. Paraizo, "Novel ways of parameterizing the history matching problem", SPE Latin American and Caribbean Petroleum Engineering

Conference held in Rio de Janeiro, Brazil, 2005

[40] F.J. Kucuk, and L. Ayestaran, “Analysis of simultaneously measured pressure and sandface flow rate in transient well testing”, JPT, 2 ,323-343 (1985)

[41] F.J. Kuchuk, “Application of convolution and deconvolution to transient well tests”, SPEF E, 12, 375-394 (1986)

[42] F.J. Kuchuk, F. Hollaender, I.M. Gok, and M. Onur,“Decline Curves from Deconvolution of Pressure and Flow-Rate Measurements for Production Optimization and Prediction ” SPE Annual Technical Conference and Exhibition, Dallas, Texas, 9-12 October 2005

[43] G.J Richardson, J.F. Roux, P. Quinn, S.D. Harker, and L.E. Sides, "Characterization of Low-Permeability Gas Condensate Behaviour by Intensive Reservoir Monitoring", Paper SPE 77618 presented at the SPE Annual Technical Conference and Exhibition, San Antonio, Texas, 2 October2002

[44] H.M. Froa, and W. Destro, "Reliability Evolution of Permanent Downhole Gauges for Campos Basin Subsea Wells: A 10-year case study", SPE Annual Technical Conference and Exhibition, San Antonio, Texas, USA, 24-27 September 2006

[45] H.Z. Meng, E.A. Proano, I.M. Buhidma, J.M. Mach,” Production Systems Analysis of Vertically Fractured Wells”, Paper SPE 10842 presented at the SPE/DOE Unconvential Gas Recovery Symposium, Pittsburgh, Pennsylvania, May16-18, 1982.

[46] Henk Kool, Mehdi Azari: "Testing of Gas Condensate Reservoirs — Sampling, Test Design and Analysis", SPE Asia Pacific Oil and Gas Conference and Exhibition, Jakarta, Indonesia, 17–19 April 2001

[47] H.J.Jr. Ramey, “Short-Time Well Test Data Interpretation in the Presence of Skin Effect and Wellbore Storage”, Journal of Petroleum Technology, Volume 22, Number 1, 97-104, (1970)

[48] H.J.Jr. Ramey, “Non-Darcy Flow and Wellbore Storage Effects in Pressure Build-Up and Drawdown of Gas Wells”, Journal of Petroleum Technology, Volume 17, Number 2, 223-233, (1965)

[49] H.J.Jr. Ramey, “Advances in Practical well-test analysis”, SPE JPT, 44(6), 650-659 (1992)

[50] James L. Buchwalter, Ray E. Calvert, Gemini Solutions; Colin S. McKay, Stephen J. Thompson, “Maximizing Profitability in Reservoirs Using New Technologies For Continuous Downhole Pressure Systems”, SPE Annual Technical Conference and Exhibition, Dallas, Texas, 1-4 October 2000

[51] Javier Ballinas, “Evaluation and Control of Drilling, Completion and Workover Events with Permanent Downhole Monitoring : Applications to Maximize Production and Optimize Reservoir Management”, SPE International Petroleum Conference and Exhibition in Mexico, Villahermosa, Mexico, 10-12 February 2002

[52] J.R. Jargon, and H.K. Van Poolen, “Unit response function from varying field data”, JPT, 965, Trans, AIME, 234-251(1965)

[53] K.C Khong, "Permanent Downhole Gauge Data Interpretation", MS report, Stanford University, Palo Alto, California, (2001)

[54] Kikani Jitendra, and He Meiqing, ”Multi-resolution Analysis of Long-Term Pressure Transient Data Using Wavelet Methods”, paper SPE 48966 presented at the 1998 SPE Annual Technical Conference and Exhibition held in New Orleans, Louisiana

[55] K.H.Coats., et al, “Determination of Aquifer Influence Functions From Field Data”, Trans, AIME, 231, 1417, 1964

[56] K. Kuchuk, F. Hollaender, I.M. Gok, M. Onur, "Decline Curves from Deconvolution of Pressure and Flow-Rate Measurements for Production Optimization and Prediction", Paper SPE 96002 presented at the SPE Annual Technical Conference and Exhibition, Dallas, 9–12 October 2005

[57] K. Sun, M.R. Konopczynski, "Using Downhole Real-Time Data to Estimate Zonal Production in a Commingled-Multiple-Zones Intelligent System", Paper SPE 102743 presented at the SPE Annual Technical Conference and Exhibition held in San Antonio, Texas,U.S.A, 24-27September2006

[58] L. Breiman, J.H. Friedman, "Estimating Optimal Transformations for Multiple Regression and Correlation", Journal of American Statistical Association, 80(391), 580–619 (1985)

[59] Lee, J.H., "Estimating Time-Dependent Reservoir Properties by Analyzing Long-Term Pressure Data". MS report, Stanford University, Palo Alto, California (2003)

[60] L.B. Ouyang, and Kikani, J. "Improving Permanent Downhole Gauge (PDG) Data Processing via Wavelet Analysis". Paper SPE 78290 presented at the European Petroleum Conference, Aberdeen, 29–31 October 2002.

[61] L.B. Ouyang and Ramzy Sawiris: "Production and injection Profiling: A novel Application of Permanent Downhole Pressure Gauges", Paper SPE 84399, presented at the SPE Annual Technical Conference and Exhibition held in Denver Colorado USA, 5-8 October, 2003

[62] L. Guan, L. Li.U., "Wavelets in petroleum Industry: Past, Present and Future", SPE Annual Technical Conference and Exhibition, Houston, Texas, 26-29 September 2004

[63] L.G.Thompson, and A.C. Reynolds, "Analysis of variable-rate well test pressure data using Duhamel's principle", SPEFE, 10, 453-472 (1986)

[64] M. Cinar; D. Ilk, M. Onur, and P.P Valko, and T.A. Blasingame, "A Comparative Study of Recent Robust Deconvolution Algorithms for Well-Test and Production-Data Analysis", SPE Annual Technical Conference and Exhibition, San Antonio, Texas, USA, 24-27 September 2006

[65] M. M. Faskhoodi, Kelkar and Assocs, "Using the Low-Frequency Observation Data in Automatic History Matching", Paper SPE 105252 presented at the 15 middle East Oil & Gas Show and conference held in Bahrain International Exhibition Center, Kingdom of Bahrain, 11-14 March 2007

[66] Magnus Nnadi, Michael Onyekonwu, "Numerical Welltest Analysis", Nigeria Annual International Conference and Exhibition, Abuja, Nigeria, 2-4 August 2004

[67] M.Mc Cracken and D.Chorneyko: "Rate Allocation Using Permanent Downhole Pressure", Paper SPE 103222, presented at the SPE Annual Technical Conference and Exhibition held in San Antonio, Texas, 24-27 September, 2006

[68] M.M. Kamal, Y. Pan, J.L. Landa, and O.O. Thomas, "Numerical Well Testing: A Method To Use Transient Testing Results in Reservoir Simulation", SPE Annual Technical Conference and Exhibition, Dallas, Texas, 9-12 October 2005

[69] M. M. Levitan, SPE, BP America, "Practical Application of Pressure/Rate Deconvolution to Analysis of Real Well Tests", SPE Reservoir Evaluation & Engineering, Volume 8, Number 2, 113-121 (2005)

[70] M.M. Levitan, "Practical Application of Pressure/Rate Deconvolution to Analysis of Real Well Tests", SPE Reservoir Evaluation & Engineering, 8 (2), 113–121 (2005)

[71] Michael M. Levitan, Gary E. Crawford, and Andrew Hardwick, "Practical Considerations for Pressure-Rate Deconvolution of Well-Test Data", SPE Journal, Volume 11, Number 1, 35-47, 2006

[72] M.M. Levitan, SPE, BP plc, "Deconvolution of Multi-Well Test Data", SPE Annual Technical Conference and Exhibition, San Antonio, Texas, USA, 24-27 September 2006

[73] Michael M. Levitan, SPE, BP plc, "Deconvolution of Multiwell Test Data", SPE Journal, Volume 12, Number 4, 420-428 (2007)

[74] M. Nomura, "Processing and Interpretation of Pressure Transient Data from

Permanent Downhole Gauges", PhD dissertation, Stanford University, Palo Alto, California (2006)

[75] M. Onur, D. Ilk, P.P Valko, and P.S. Hegeman, “An Investigation of Recent Deconvolution Methods for Well-Test Data Analysis”, SPE Journal, Volume 13, Number 2, 226-247 (2008)

[76] M. Parker, Kerr McGee; R.N. Bradford, and C. Corbett, R.N. Heim, C.L. Isakson, S.S. Broome, and E. Proano, “Using Real-Time Pressure Data for Well Placement Planning”, SPE/DOE Symposium on Improved Oil Recovery, Tulsa, Oklahoma, USA, 22-26 April 2006

[77] M.P. Tibold, S. Simonian, M. Chawla, M. Akbar, ,” Well Testing with a Permanent Monitoring System”, SPE Annual Technical Conference and Exhibition, Dallas, Texas, 1-4 October 2000

[78] M.S. Laws, T.Matsuura, H.F. Soek, and P.M. O'Dell, and T.J.F. Kinsella, Schlumberger, "Permanent Downhole Pressure Gauges Help Underpin Feasibility of Miscible Gas Flood", Paper SPE93553 present at the 14 SPE Middle East Oil&Gas show and conference held in Bahrain international exhibition centre, Bahrain,12-15 March 2005

[79] M.Y. Soliman, Ansah, and B. Manda, "Application of Wavelet Transform to Analysis of Pressure Tests", SPE Annual Technical Conference and Exhibition, New Orleans, Louisiana, 30 September-3 October 2001

[80] Muhammad Shafiq; Omar Al-Faraj, Adnan A. Al-Kanaan, and Bandar H. Al-Malki, “First High Pressure and High Temperature Digital Electric Intellitite Welded Permanent Down Hole Monitoring System for Gas Wells”, SPE Saudi Arabia Section Technical Symposium, Al-Khobar, Saudi Arabia, 10-12 May 2008

[81] M. Weaver, T. Kragas, SPE, J. Burman, D. Copeland, B. Phillips, and R. Seagraves, “Installation and Application of Permanent Downhole Optical Pressure/Temperature Gauges and Distributed Temperature Sensing in Producing Deepwater Wells at Marco Polo”, SPE Annual Technical Conference and Exhibition, Dallas, Texas, 9-12 October 2005

[82] O. Bahabanian, D. Ilk, N. Hosseinpour-Zonoozi, and T.A. Blasingame, “Explicit Deconvolution of Wellbore Storage Distorted Well Test Data”, SPE Annual Technical Conference and Exhibition, San Antonio, Texas, USA, 24-27 September 2006

[83] Obinna Duru, SPE, and Roland N. Horne, “Modeling Reservoir Temperature Transients and Matching to Permanent Downhole Gauge Data for Reservoir Parameter

Estimation", SPE Annual Technical Conference and Exhibition, Denver, Colorado, USA, 21-24 September 2008

[84] Ochi I. Achinivu, Zhuoyi Li, D. Zhu, and A.D. Hill, "An Interpretation Method of Downhole Temperature and Pressure Data for Flow Profiles in Gas Wells (Russian)", SPE Russian Oil and Gas Technical Conference and Exhibition, Moscow, Russia, 28-30 October 2008

[85] O.O.A. Thomas, "The Data as the Model: Interpreting Permanent Downhole Gauge Data without Knowing the Reservoir Model", MS report, Stanford University, Palo Alto, California (2002)

[86] P.M. Ribeiro, and A.P. Pires, "Use of Wavelet Transform in PDG Data Treatment", Paper SPE 100719 presented at the SPE Annual Technical Conference and Exhibition, San Antonio, Texas, 24–27 September2006

[87] P.M. Ribeiro, A.P. Pires, E.A.P. Oliveira, and J.G. Ferroni, "Use of Wavelet Transform in Pressure-Data Treatment", SPE Production & Operations, Volume 23, Number 1, 24-31 (2008)

[88] P.S. Fair, J.F. Simmons, "Novel Well Testing Applications of Laplace Transform Deconvolution", SPE Annual Technical Conference and Exhibition, Washington, D.C., 4-7 October 1992

[89] Rai Himansu, Roland N.Horne, "Analyzing Simultaneous Rate and Pressure Data From Permanent Downhole Gauges", SPE Annual Technical Conference and Exhibition, Anaheim, California,U.S.A, 11-14 November 2007

[90] Rai Himansu, "Analyzing Rate Data from Permanent Downhole Gauges", MS report, Stanford University, Palo Alto, California, (2005)

[91] R.C. Jr. Earlougher, "Advances in well test analysis", Monograph Series, Richardson, Texas, (1977)

[92] R.G. Agarwal, "A New Method to Account for producing time effects when drawdown type curves are used to analyze pressure buildup and other test data", SPE Annual Technical Conference and Exhibition, Dallas, Texas, 21-24 September 1980

[93] Risi Jadesola Omotosho: "Permanent Downhole Sensors in Today's Petroleum Industry", MS report, the University of Texas at Austin, (2004)

[94] Roland N.Horne: "Modern Well Test Analysis—A Computer-Aided Approach (Second Edition)", Petroway, (1998)

[95] Roland N. Horne，"Listening to the Reservoir—Interpreting Data From Permanent Downhole Gauges", Journal of Petroleum Technology, Volume 59, Number 12, 78-86

(2007)

[96] Roland N. Horne, SPE, Stanford University, "Advances in Computer-Aided well-Test Interpretation", Paper SPE 24731, Presented at the SPE annual technical conference and exhibition held in Washington, 4-7 Oct, 1992

[97] Sammy Haddad, Eduardo Proano, and Yogesh Patel:"A method to Diagnose Depletion, Skin,kh, and Drive Methanism Effects Using Reservoir Monitoring Data", Paper SPE 90032, presented at SPE Annual Technical Conference and Exhibition held in Houston, Texas,U.S.A, 26-29 September 2004

[98] Sanghui Ahn, SPE, and Roland N. Horne," Analysis of Permanent Downhole Gauge Data by Cointerpretation of Simultaneous Pressure and Flow Rate Signals", SPE Annual Technical Conference and Exhibition, Denver, Colorado, USA, 21-24 September 2008

[99] Shepherd, C.E., Neve, Patrick, Wilson, D.C., "Use and Application of Permanent Downhole Pressure Gauges in the Balmoral Field and Satellite Structures", SPE Production Engineering, Volume 6, Number 3, 271-276 (1991)

[100] S. Maharaj, J. R. Jones, H. M. Mackow, D. P. Lachance, "Design and Interpretation of Long-Term Tests in the Mahogany Field, Offshore Trinidad", SPE/CERI Gas Technology Symposium, Calgary, Alberta, Canada, 3-5 April 2000

[101] S. Olsen, and J.E. Nordtvedt,. Experience from the Use of Automatic Well-Test Analysis. Paper SPE 102920 presented at the SPE Annual Technical Conference and Exhibition, San Antonio, Texas, 24–27 September 2006

[102] S. Olsen, and J.E. Nordtvedt, "Improved Wavelet Filtering and compression of production data", Paper SPE 96800 presented at offshore Europe 2005 held in Aberdeen, Scotland,U.K.,6-9September 2005

[103] S. Olsen, and J.E. Nordtvedt, "Automatic filtering and Monitoring of Real-Time Reservoir and Production Data", Paper SPE 96553 presented at the SPE Annual Technical Conference and Exhibition, Dallas, 9–12 October 2005

[104] Suwat Athichanagorn, Roland N. Horne, Jitendra Kikani, "Processing and Interpretation of Long-Term Data Acquired From Permanent Pressure Gauges", SPE Reservoir Evaluation & Engineering, 384-391 (2002)

[105] Suwat Athichanagorn, Roland N. Horne, Jitendra Kikani, "Processing and Interpretation of Long-Term Data Acquired From Permanent Pressure Gauges", SPE Annual Technical Conference and Exhibition, 3-6 October 1999, Houston, Texas

[106] Suwat Athichanagorn, "Development of an Interpretation Methodology for

Long-Term Pressure Data from Permanent Downhole Gauges", PhD dissertation, Stanford University, Palo Alto, California (1999)

[107] Tim Whittle, and Alain Gringarten, "The Determination of Minimum Tested Volume from the Deconvolution of Well Test Pressure Transients", SPE Annual Technical Conference and Exhibition, Denver, Colorado, USA, 21-24 September 2008

[108] T.S. Hutchinson and V.J. Sikora, "A generalized water drive analysis", Trans, AIME, 216, 169 (1959)

[109] T. Unneland, Y. Manin, and F. Kuchuk, "Permanent Gauge Pressure and Rate Measurements for Reservoir Description and Well Monitoring: Field Cases", SPE Reservoir Evaluation & Engineering,1 (3), 224–230. (1998)

[110] T. Unneland, and T. Haugland, "PDGs Used in Reservoir Management of Complex North Sea Oil Fields" SPEPF 9 (3), 195–203, Paper SPE 26781-PA, (1994)

[111] T. von Schroeter, F. Hollaender, and A.C. Gringarten, " Deconvolution of Well-Test Data as a Nonlinear Total Least-Squares Problem", SPE Journal, 9 (4), 375–390, (2004.)

[112] T. von Schroeter, F. Hollaender, and A.C. Gringarten, "Analysis of Well Test Data from Permanent Downhole Gauges by Deconvolution", SPE Annual Technical Conference and Exhibition, San Antonio, Texas, 29 September-2 October 2002

[113] T. von Schroeter, and A.C. Gringarten, "Superposition Principle and Reciprocity for Pressure-Rate Deconvolution of Data From Interfering Wells", SPE Annual Technical Conference and Exhibition, Anaheim, California, U.S.A., 11-14 November 2007

[114] Tor K. Kragas, F.X. Bostick III, Christopher Mayeu, Daniel L. Gysling; Alex M. van der Spek, "Downhole Fiber-Optic Flowmeter: Design, Operating Principle, Testing, and Field Installations", SPE Production & Facilities, Volume 18, Number 4, 257-268 (2003)

[115] Tor K. Kragas; Bill F. Turnbull, Michael J. Francis," Permanent Fiber-Optic Monitoring at Northstar: Pressure/Temperature System and Data Overview", SPE Production & Facilities, Volume 19, Number 2, 86-93 (2004)

[116] Trond. Unneland, Statoil A/S," Permanent Downhole Gauges Used in Reservoir Management of Complex North Sea Oil Fields", SPE Production & Facilities, Volume 9, Number 3, 195-203 (1994)

[117] Valeria Matamoros, PDVSA E&P; Elena Escobar, PDVSA Intevep; Anaiza Rodriguez, PDVSA E&P," Monitoring Techniques For HP/HT Reservoirs: Furrial Field

Case", SPE Latin American and Caribbean Petroleum Engineering Conference, Port-of-Spain, Trinidad and Tobago, 27-30 April 2003

[118]W.A. Nestlerode, "The Use of Pressure Data from Permanently Installed Bottomhole Pressure Gauges", Paper SPE 590 presented at the SPE Rocky Mountain Joint Regional Meeting, Denver, 27–28 May 1963

[119] W. Mathis, Thonhauser and G. Thonhauser, "Mastering Real-time Data Quality Control-how to Measure and manage the Quality of (Rig) Sensor Data", Paper SPE 107567 presented at the SPE/IADC Middle East Drilling Technology Conference & Exhibition held in Cairo,Egypt,October,2007

[120] William D.McCain, "The Properties of Petroleum Fluids (second edition)", PennWell Publishing Company, (1990)

[121] Wavelet/Matlab, http://www.mathworks.com/products/wavelet/

[122] Yueming Cheng; W. John Lee; Duane A. McVay, "Fast Fourier Transform Based Deconvolution for Interpretation of Pressure-Transient-Test Data Dominated by Wellbore Storage", SPE Reservoir Evaluation & Engineering, Volume 8, Number 3, 224-239 (2005)

[123] Y.T. Chui, "An introduction to Wavelets", Academic Press, (1992)

[124] ZIEBEL, "Towards intelligent wells", http://www.ziebel.no/activecompletions/

[125] Zheng Shiyi, "Fighting Against NonUnique Solution Problems in Heterogeneous Reservoirs Through Numerical Well testing ", SPE Asia Pacific Oil & Gas Conference and Exhibition, Adelaide, Australia, 11-13 September 2006

[126] Zheng Shiyi and Li X.G., 2007,"Analyzing Transient Pressure from Permanent Downhole Gauge (PDG) using wavelet method", Paper SPE 107521 presented at the SPE Europec/EAGE Annual Conference and Exhibition held in London, UK, 11-14 June 2007

[127] Zheng Shiyi and Li X.G., 2007, "Transient Pressure analysis of 4D reservoir system response from permanent downhole gauge (PDG) for reservoir monitoring , testing and management", Paper SPE109112 present at the 2007 SPE Asia Pacific Oil & Gas Conference and Exhibition Held in Jakarta, Indonesia, November 2007

[128] Zheng Shiyi, P. Corbett, and G. Stewart, "Te impact of variable formation thickness on pressure transient behavior and well test permeability in Fluvial meander loop reservoirs", Paper SPE 36552 presented at the SPE Annual Technical Conference and Exhibition, Denver,6-9 Oct 1996

[129] Z. Wu, F.O. Alpak, C. Torres-Verdin, "A Quantitative Study to Assess the Value

of Pressure Data Acquired with In-Situ Permanent Sensors in Complex 3D Reservoir Models Subject to Two-Phase Fluid Flow", SPE Annual Technical Conference and Exhibition, Denver, Colorado, 5-8 October 2003